НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет Інформатики і Обчислювальної Техніки
Кафедра Обчислювальної Техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій Григорович СТІРЕНКО

«___»_____2020 р.

Дипломний проект

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»
за спеціальністю «Інженерія програмного забезпечення»
на тему: «Клієнт-серверний додаток для керування системою кінотеатру»

Виконав :
студент IV курсу, групи ІП-62
Євгеній Володимирович МІЛАШЕВСЬКИЙ _____
Керівник:
Асистент
Микита Сергійович РУЖЕВСЬКИЙ _____
Консультант з нормоконтролю:
Професор, доктор технічних наук
Валерій Павлович СІМОНЕНКО _____
Рецензент:
_____
_____ _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	ІАЛЦ 467400.000 ВП	Відомість проекту	1	
3	А4	ІАЛЦ 467400.001 ТЗ	Технічне завдання	2	
4	А4	ІАЛЦ 467400.002 ПЗ	Пояснювальна записка	50	
5	А4	ІАЛЦ 467400.003 Д1	Функціональна схема	1	
6	А4	ІАЛЦ 467400.004 Д2	Структурна схема	1	
7	А4	ІАЛЦ 467400.005 Д3	Принципова схема алгоритму покупки квитків	1	
8	А4	ІАЛЦ 467400.006 Д4	Лістинг частин програми	10	

Зм..	Фрк	№ докум	Підпис	Дата	ІАЛЦ 457400.000 ВП		
Разробив		Мілашевський Є.			*Відомість дипломного проекту*	Літер.	Арк.	Аркуші
Керівник		Ружевський М.С.				у	2	1
Рецензія						НТУУ "КПІ" ФІОТ ІП-62		
Н. Контр.		Сімоненко В.П						
Затв.								

Пояснювальна записка
до дипломного проекту
на тему: «Клієнт-серверний додаток для керування системою кінотеатру»

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут”
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень бакалавр
Напрям підготовки 6.050103 « Інженерія програмного забезпечення »

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій Григорович СТІРЕНКО
(підпис) (ініціали, прізвище)
«___» _____ 2020р.

ЗАВДАННЯ

на бакалаврську дипломну роботу студента

_____ *Євгенія Володимировича МІЛАШЕВСЬКОГО* _____

(прізвище, ім’я, по батькові)

1. Тема проекту (роботи) *Клієнт-серверний додаток для керування системою кінотеатру*

керівник проекту (роботи) *асистент Микита Сергійович РУЖЕВСЬКИЙ*,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

2. Термін здачі студентом закінченого проекту (роботи) *4 червня 2020р.*.

3. Вихідні дані до проекту (роботи) *технічна документація. теоретичні та статистичні дані*_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються) *Розгляд існуючих архітектур, огляд існуючих програмних рішень та технологій, програмна реалізація додатку.*

5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень) структурна схема системи, узагальнена схема роботи системи, блок-схема алгоритму системи.

6. Консультанта проекту (робота), з вказівкою розділів роботи, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проекту (роботи)	Строк виконання етапів проекту(роботи)	Примітки
/.	*Затвердження теми роботи*	*10.12.2019-15.12.2019*	
2.	*Вивчення та аналіз завдання*	*15.12.2019-15.03.2020*	
3.	*Розробка архітектури та загальної структури систем*	*15.03.2020-15.04.2020*	
4.	*Розробка структур окремих підсистем*	*15.04.2020-01.05.2020*	
5.	*Програмна реалізація системи*	*01.05.2020-15.05.2020*	
6.	*Оформлення пояснювальної записки*	*15.05.2020-25.05.2020*	
7.	*Захист програмного продукту*		
8.	*Передзахист*		
9.	*Захист*		

Студент-дипломник _____

(підпис)

Керівник роботи _____

(підпис)

АНОТАЦІЯ

Дипломну роботу виконано на 68 аркушах, вона містить 4 додатки та перелік посилань на використані джерела з 10 найменувань. У роботі наведено 29 рисунків та 3 таблиці.

Метою даної дипломної роботи є створення клієнт-серверного додатку для керування системою кінотеатру.

У роботі проведено аналіз існуючих типів архітектурних рішень поставленої задачі — багатошарові типи архітектур, базові відомості щодо проектування архітектури. Виконано їх порівняння з погляду ефективності розробки, стійкості до навантажень, відмовостійкості, надійності зберігання даних. Для розв’язання задачі в роботі тришарову архітектуру побудовану за принципом REST, основану на структурному шаблоні Модель-Вид-Контролер.

Ключові слова: клієнт-серверний, клієнт, сервер, архітектура, фреймворк, модель, вид представлення, відмовостійкість, дизайн, система, додаток.

АННОТАЦИЯ

Дипломную работу выполнено на 68 листах, она содержит 4 приложения и перечень ссылок на использованные источники с 10 наименований. В работе приведены 29 рисунков и 3 таблицы.

Целью данной работы является создание клиент-серверного приложения для управления системой кинотеатра.

В работе проведен анализ существующих типов архитектурных решений поставленной задачи - многослойные типы архитектур, базовые сведения по проектированию архитектуры. Выполнено их сравнение с точки зрения эффективности разработки, устойчивости к нагрузкам, отказоустойчивости, надежности хранения данных. Для решения задачи в работе трехслойную архитектуру построенную по принципу REST, основанную на структурном шаблоне Модель-Вид-Контроллер.

Ключевые слова: клиент-серверный клиент, сервер, архитектура, фреймворк, модель, вид представления, отказоустойчивость, дизайн, система, приложения.

ABSTRACT

The thesis is performed on 68 sheets, it contains 4 appendices and a list of references to the sources used with 10 titles. The paper contains 29 figures and 3 tables.

The purpose of this thesis is to create a client-server application for managing the cinema system.

The analysis of the existing types of architectural solutions of the set task - multilayer types of architectures, basic information on architecture design is carried out in the work. Their comparison is performed in terms of development efficiency, load resistance, fault tolerance, reliability of data storage. To solve the problem, the three-layer architecture is built on the principle of REST, based on the structural template Model-View-Controller.

Keywords: client-server, client, server, architecture, framework, model, type of representation, fault tolerance, design, system, application.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

API - Application Programming Interface, інтерфейс програмування

HTTP (HyperText Transfer Protocol) протокол передачі гіпертексту

HTML (Hypertext Markup Language) Мова розмітки гіпертекстових документів.

REST (Representational State Transfer) Передача репрезентативного стану.

SOAP (Simple Object Access Protocol) Простий протокол доступу до об'єктів.

XML (eXtensible Markup Language) Розширювана мова розмітки.

JVM (Java Virtual Machine) віртуальна машина Java

JRE (Java Runtime Environment) середовище виконання Java

SQL (Structured Query Language) структурована мова запитів

СКБД Система керування базами даних

БД База даних

NoSQL (Not Only SQL) не тільки SQL

MVC (Model View Controller) модель-вид-представлення

ВСТУП

Ще з давніх пір мистецтво кіно вабило людину. Перші кінотеатри з'явились ще за часі дикого заходу. Тоді це було щось нове, незвичне сьогодні ж кіно увійшло в наше життя і стало його частиною.

Велика частка суспільства полюбляє провести годинку-другу за переглядом цікавої історії. Дивитися ж кіно на великому екрані – саме задоволення. Саме тому сьогодні кінотеатри такі популярні. А як відомо попит породжує пропозицію.

Метою мого проекту було створити сайт для невеликого кінотеатру, з можливістю подальшого розвитку та розширення – комерційний проект який буде цікавий замовнику.

Зважаючи на те що така система може бути схильна до великого навантаження, потрібно розробити систему яка зможе опрацьовувати запити сотень людей.

Вимоги до системи диктуються споживачем тому система мусить мати надзвичайно дружелюбний інтерфейс та достатню швидкість відгуку. Враховуючи сучасні реалії потрібно памятати про можливість використання системи на різних платформах.

Результатом даної роботи буде повноцінний додаток для керування системою невеликого кінотеатру.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ АРХІТЕКТУР

1.1 Визначення

Архітектура інформаційної системи –це специфікація, що детально описує взаємодію набору стандартів програмних та апаратних технологій для формування комп'ютерної системи або платформи. Коротше кажучи, комп'ютерна архітектура стосується того, як створена комп'ютерна система та з якими технологіями вона сумісна.

Клієнт-серверна архітектура – це обчислювальна архітектура виробник / споживач, де сервер виступає як виробник, а клієнт - як споживач. Сервер розміщує і надає клієнту послуги високого класу, що вимагають високих обчислень, на вимогу. Ці сервіси можуть включати доступ до додатків, зберігання даних, обмін файлами, доступ до принтера та / або прямий доступ до необмеженої обчислювальної потужності сервера.[2, 25-30]

Сервер – це комп’ютер чи програма, що надає певні послуги іншим програмам і обробляє повідомлення отримані від клієнтів.

Клієнт –це кінцевий споживач послуги, або запитувач послуги в системі типу клієнт / сервер. Клієнт найчастіше розташований на системі чи комп’ютері, до якого можна отримати доступ через мережу. Цей термін вперше застосовувався для пристроїв, які не могли запускати власні програми, і були підключені до віддалених комп'ютерів через мережу. Їх називали «тупими терміналами».

1.2 Опис моделі клієнт-сервер

Обчислювальна модель клієнт-сервер займає домінуюче місце серед методів розподілених обчислень. Модель дозволяє розділяти функціонал і обчислювальну навантаження між клієнтськими додатками (замовниками послуг) і серверними додатками (постачальниками послуг)., зображення схеми такої системи показано на рисунку 1.1. Такий поділ дозволяє позбавити персональні комп’ютери певних недоліків, як-то зменшення обчислювальної потужності девайсів.

Як правило, користувачам комп’ютерів потрібна висока обчислювальна

потужність і широкі можливості та корисні властивості персональних комп'ютерів. Обробка даних може бути розподілена по різному, залежно від використаного програмного забезпечення. Сервер приймає запит від клієнта і повертає йому результат.

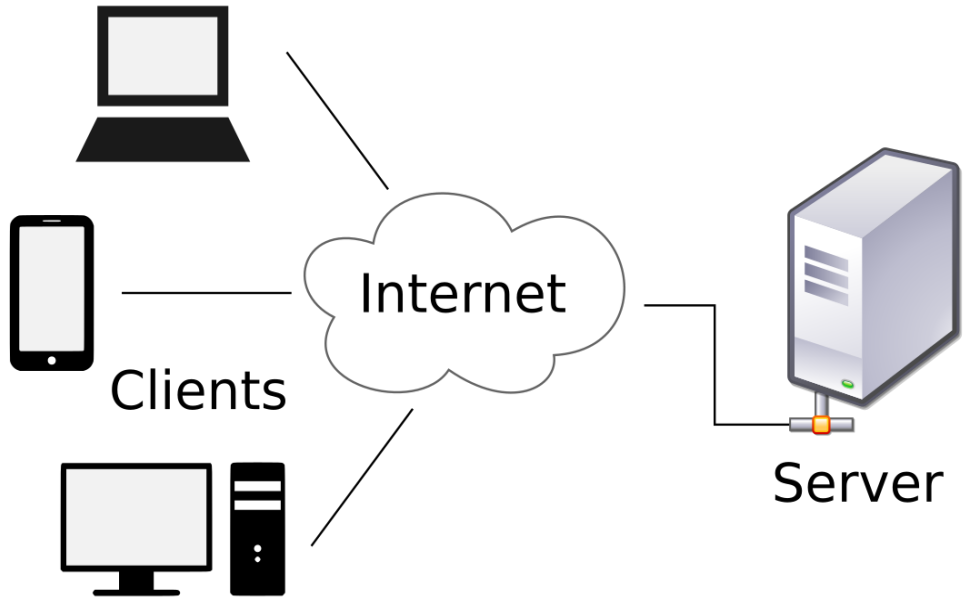


Рисунок 1.1 – Архітектура клієнт-сервер

1.3 Класи клієнт-серверних додатків

Клієнт-серверна архітектура додатків пропонує по-різному розділити виконувану роботу між клієнтом та сервером.. Частина виконуваних операцій і об'єм даних що були передані через комп'ютерну мережу залежать від природи інформації, що зберігається в базі даних, підтримуваних типів додатків, доступності технічного, яке працюватиме паралельно, а також від способу використання даних.

Схеми основних класів клієнт-серверних додатків показані на рисунках 1.2, 1.3, 1.4, 1.5.

1.3.1 Обробка даних на стороні хоста

Дана схема не є справжнім додатком клієнт-сервер. Термін «хост» зазвичай використовується в випадку коли існують дві комп'ютерні системи, з'єднані модемами та лініями телефонного зв'язку. Таким чином, система що зберігає

данні та виконує всі обчислення називається хостом, а інтерфейс користувача має вигляд примітивного терміналу. Всі сервери вважаються хостами, проте не всі хости є серверами.

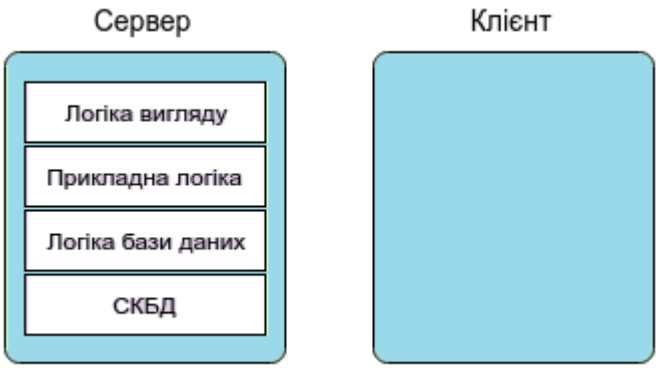


Рисунок 1.2 – Обробка даних на стороні хоста

Але навіть якщо клієнт використовує комп'ютер для роботи клієнт-серверного додатку, основні задачі цього комп’ютера обмежуються емуляцією терміналу. Така архітектура є доцільною, коли потужностей які мають клієнти недостатньо для обробки даних. Як правило, сервер-хост набагато потужніший аніж девайси користувачів .

1.3.2 Обробка даних на стороні сервера

Найпростішим та найпоширенішим класом клієнт-серверних застосунків є схема, в якій сторона клієнта відповідає лише за графічне відображення додатку, тоді як вся обробка даних здійснюється на стороні сервераВ такому випадку навантаження на серверну частину знижується, особливо у випадках, коли логіка вигляду потребує багато системних ресурсів для обробки.

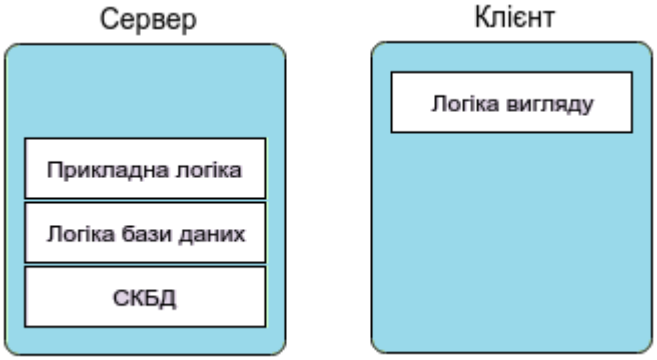


Рисунок 1.3 – Обробка даних на стороні сервера

1.3.3 Обробка даних на стороні клієнта

Дана архітектурна схема демонструє інакший підхід: майже вся логіка обробки даних здійснюється використовуючи потужності клієнта, за винятком логіки що повязана зі зберіганням данних. Як правило, складні функції для роботи з базою даних виконуються на стороні клієнта

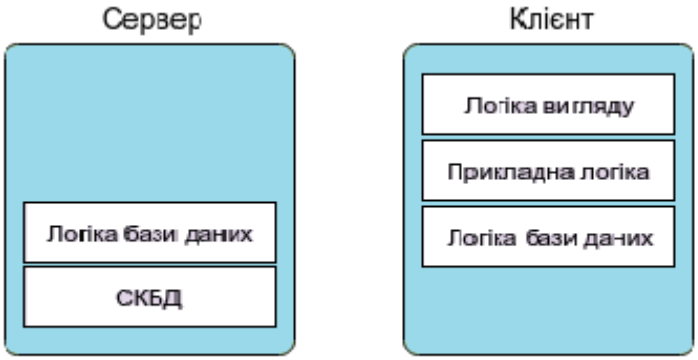


Рисунок 1.4 – Обробка даних на базі клієнта

1.3.4 Спільна обробка даних

У архітектурній схемі обробка даних налаштована таким чином, щоб використовувати наявні потужності як сервера так і клієнта, а також рівномірно розприділити дані.

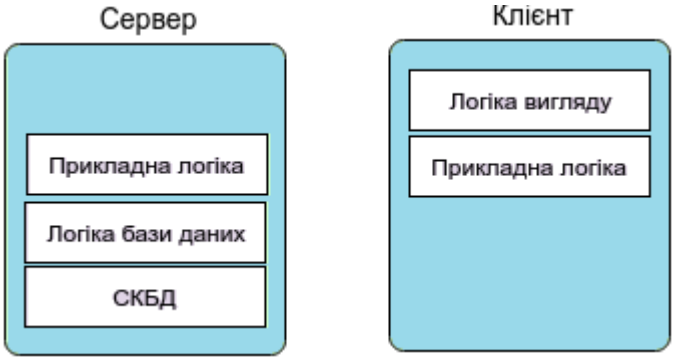


Рисунок 1.5 – Спільна обробка даних

Такі системи набагато складніші в інсталяції і підтримці, проте вони дозволяють підвищити показники продуктивності, адже використовують всі доступні ресурси.

Рисунок 1.4 та рисунок 1.5 відповідають конфігураціям, в яких більша частина роботи виконується на стороні клієнта, такі типи клієнтів називають

“товстими”. Товстий або Rich-клієнт в архітектурі клієнт-сервер – це програма з розширеними функціональними характеристиками. У випадку використання «товстого» клієнта, сервер використовується як сховище даних, а вся робота переноситься на сторону клієнта.

1.4 Архітектура серверної частини

1.4.1 Архітектура, як поняття

Коли мова йде про поняття архітектура, зазвичай визначень більш ніж достатньо. Існують навіть деякі веб-сайти, які збирають такі визначення, ось декілька з них:

Архітектура - це базова організація системи, втілена в її компонентах, їхні стосунки між собою та з оточенням, а також принципи, що визначають проектування і розвиток системи [1]

Архітектура - це набір значущих рішень щодо організації системи програмного забезпечення, набір структурних елементів та їхніх інтерфейсів, за допомогою яких компонується система, разом з їх поведінкою, обумовленим у взаємодії між цими елементами, компонування елементів в поступово укрупнюються підсистеми, а також стиль архітектури який направляє цю організацію - елементи та їх інтерфейси, взаємодії і компоновку. [3, ст 125-130]

Архітектура - це структура організації та пов'язане з нею поводження системи. Архітектуру можна рекурсивно розібрати на частини, які взаємодіють за допомогою інтерфейсів, зв'язку, які з'єднують частини, і умови складання частин. Частини, які взаємодіють через інтерфейси, включають класи, компоненти і підсистеми. Увага у цьому розділі буде зосередженна на розшарованій архітектурі та її можливостях і способах застосування.

1.4.2 Розбиття на шари

Багатошарова архітектура - це найпростіша форма архітектурного шаблону програмного забезпечення.. На Рисунку 1.6 наведено приклад додатка бобудованого за принципом багатошаровості.

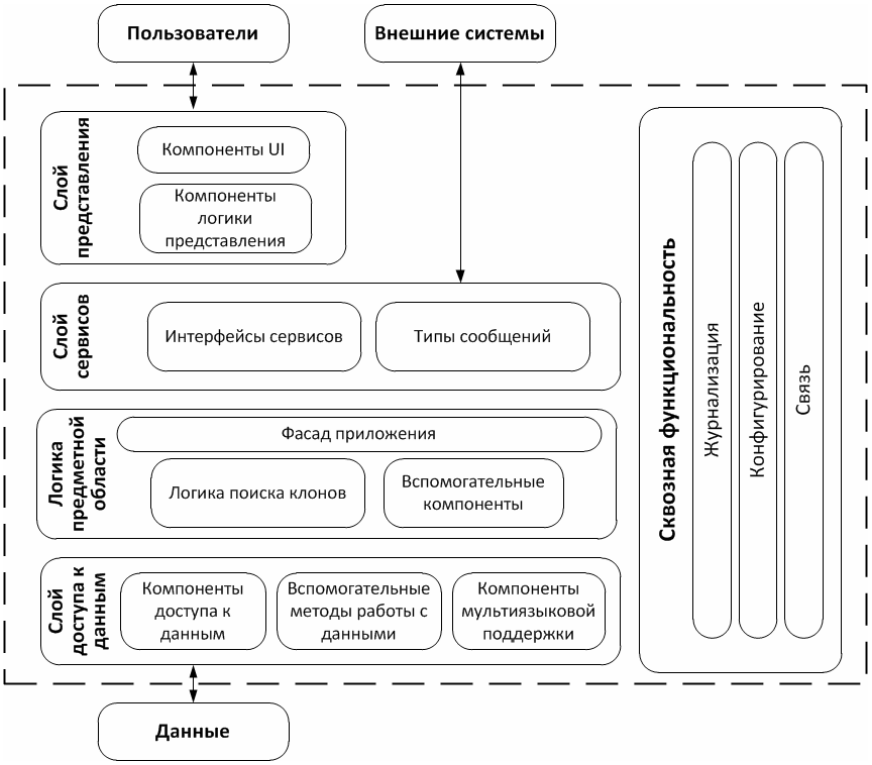


Рисунок 1.6 – Розділення додатку на шари

Якщо ви збираєтеся створити рудиментарний додаток, коли кількість користувачів дуже низька (<100–200), і ви впевнені, що після переходу в режим активної роботи не буде занадто багато змін вимог, це найкращий зразок архітектури програмного забезпечення.

Вартість реалізації цього шаблону архітектури мінімальна порівняно з іншими шаблонами.

Далі наведено плюси аналізу шаруватої структури архітектури.

Плюси

- Легкий доступ до кожного окремого шару для реалізації алгоритмів обробки інформації.
- Легко проводити зміни та підтримувати існуюче рішення, адже кожен компонент належить окремому шару додатку.

Мінуси

- Хоча зміни в певному шарі можуть бути внесені, це непросто, оскільки додаток є єдиною одиницею. Також зчеплення між шарами, як правило, ускладнює обслуговування системи. Також це ускладнює масштабування.
- Додаток повинен бути розгорнутий як єдиний блок, таким чином, зміна

певного шару означає, що вся система повинна бути перерозподілена.

• Чим система більша, тим більше ресурсів потрібно для запитів, щоб пройти через декілька шарів і, таким чином, це викликає проблеми з продуктивністю.

1.4.3 Двошарова архітектура

Дворівнева архітектура - це архітектура програмного забезпечення, в якій презентаційний рівень або інтерфейс працює на клієнті, а рівень даних або структура даних зберігаються на сервері. Поділ цих двох компонентів на різні місця представляє дворівневу архітектуру, на відміну від однорівневої архітектури. Інші види багаторівневої архітектури додають додаткових шарів у розподіленому дизайні програмного забезпечення.

Схема роботи дворівневої архітектури продемонстрована на рисунку 1.7.

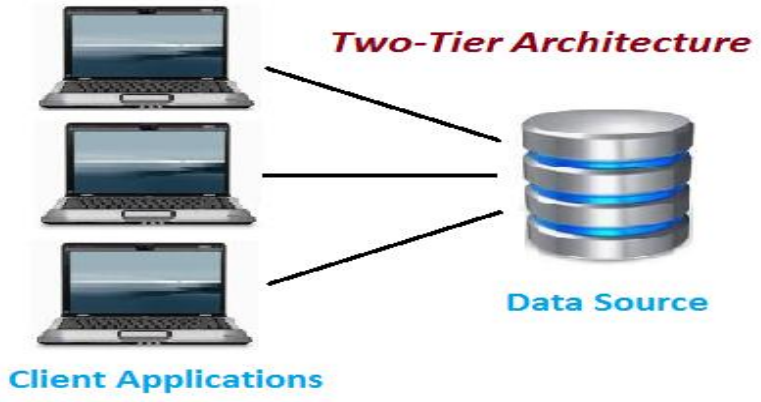


Рисунок 1.7 – Двошарова архітектура

Таблиця 1.1 – Переваги та недоліки двошарової архітектури

Переваги	***Недоліки***
Програми можна легко розробити завдяки простоті	Дворівнева модель не має масштабованості, оскільки вона підтримує лише обмежену кількість користувачів

Переваги	*Недоліки*
Максимальне задоволення користувачів отримують завдяки точному та швидкому прототипуванню програм через надійні інструменти	Більшість застосувань, які використовуються для взаємодії, залежать від структури бази даних, яка створює проблему при перепроектуванні, оскільки вони тісні з переважаючою структурою
Сервер бази даних та бізнес-логіка фізично близькі, що забезпечує більш високу продуктивність	Оскільки клієнт опрацьовує більшу частину логіки програми, виникають проблеми з контролем версії програмного забезпечення та повторним розповсюдженням нових версій.

1.4.4 Тришарова архітектура

Трирівнева архітектура - це архітектура клієнт-сервер, в якій функціональна логіка процесу, доступ до даних, зберігання даних у комп’ютері та користувальницький інтерфейс розробляються та підтримуються як незалежні модулі на окремих платформах.

Трирівнева архітектура - це модель дизайну програмного забезпечення та налагоджена архітектура програмного забезпечення.

Детальний опис шарів наведено у таблиці 1.2.

Трирівнева архітектура дозволяє модернізувати або замінити будь-який з трьох ярусів самостійно.

Інтерфейс користувача реалізований на настільному ПК та використовує стандартний графічний інтерфейс користувача з різними модулями, що працюють на сервері.

Система управління реляційними базами даних на сервері баз даних містить логіку зберігання даних комп'ютера.

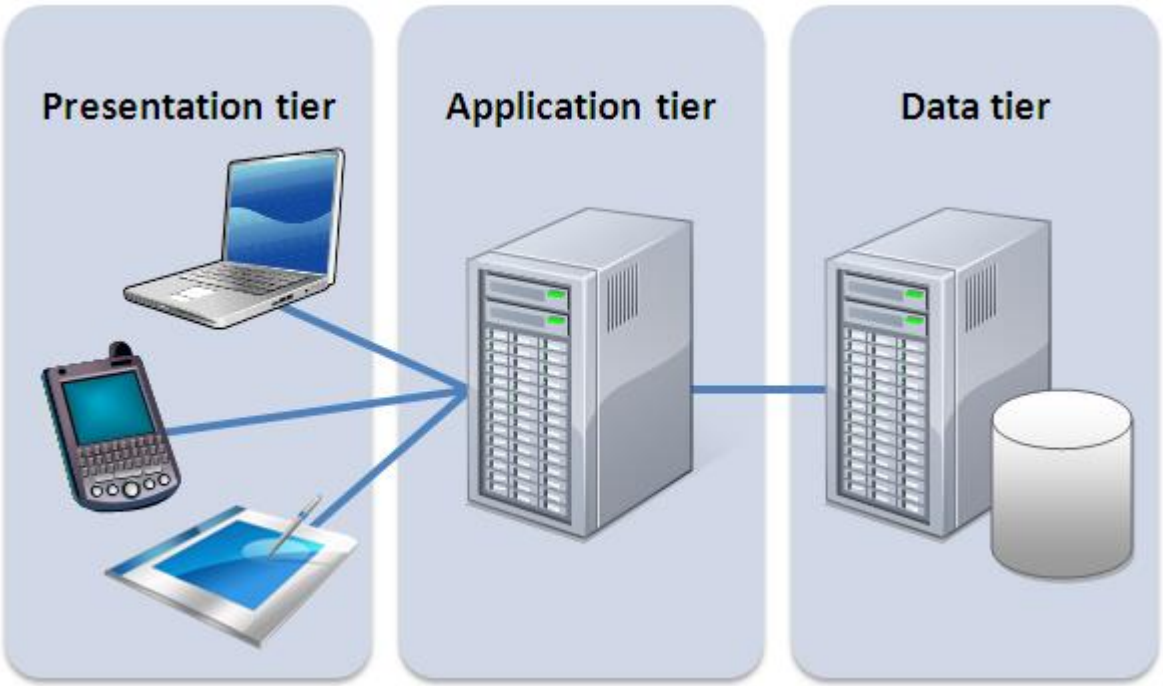


Рисунок 1.8 – Трирівнева архітектура

Оскільки ці три шари не є фізичними, а логічними за своєю суттю, вони можуть працювати на різних серверах так само як в локальних рішеннях, або ж у програмному забезпеченні (SaaS)

Таблиця 1.2 – Функції шарів тришарової архітектури

Шар	***Функції***
Рівень презентації	Займає верхній рівень і відображає інформацію, пов'язану з послугами, доступними на веб-сайті у вигляді графічного інтерфейсу користувача (GUI). Він становить передній шар програми та інтерфейс, з яким кінцеві користувачі взаємодіють через веб-додаток. Цей рівень, як правило, побудований на структурах веб-розробки, таких як CSS або JavaScript, і спілкується з іншими ярусами, надсилаючи результати в браузер та інші яруси в мережі через дзвінки API.

Шар	*Функції*
Рівень програми	Цей рівень - також його називають середнім рівнем, логічним рівнем, діловою логікою або логічним рівнем. Він контролює основні функціональні можливості програми, здійснюючи детальну обробку, і зазвичай кодується мовами програмування, такими як Python, Java, C ++, .NET тощо.
Рівень даних	Дані цього рівня зберігаються незалежно від серверів прикладних програм та бізнес-логіки, а також керуються ними та доступними програмами, такими як MongoDB, Oracle, MySQL та Microsoft SQL Server

Трирівнева архітектура має свої переваги та недоліки.

Переваги архітектури:

- розробник може зосередитись лише на всій структурі одного з шарів;
- розробник може легко використовувати нову реалізацію для заміни початкового рівня реалізації;
- полегшується повторне використання логіки кожного шару.
- висока безпека. Клієнт може отримати доступ до рівня даних лише через логічний рівень, зменшуючи точку входу і захищаючи багато небезпечних функцій системи.
- структура проекту більш чітка, а отже і більш чіткий розподіл праці, що сприяє подальшому обслуговуванню та модернізації

Недоліки архітектури:

- знижується продуктивність системи. Це само собою зрозуміло. Без багаторівневої структури багато підприємств могли отримати доступ до бази даних безпосередньо для отримання даних, а тепер їм доводиться робити це через середній рівень.
- іноді викликаються каскадні зміни. Цей вид модифікації особливо часто

відбивається в напрямку зверху вниз. Якщо вам потрібно додати функцію в шарі презентації, можливо, вам доведеться додати код до відповідного рівня бізнес-логіки та рівня доступу до даних

- збільшується кількість коду

1.4.5 Багаторівнева архітектура

Багаторівнева програма - це будь-яка програма, розроблена та розподілена між більш ніж одним шаром. Він логічно розділяє різні застосовні, операційні шари. Кількість шарів змінюється залежно від вимог бізнесу та додатків, але архітектура, що найчастіше використовується, є трирівневою. Будь-яка програма, яка залежить від або використовує програмне забезпечення середнього програмного забезпечення, відома як багаторівнева програма. Багаторівнева програма також відома як багатоярусна програма або n-ярусна програма.[12]

Багаторівнева програма використовується для поділу корпоративного додатку на два або більше компонентів, які можуть бути окремо розроблені та виконані.

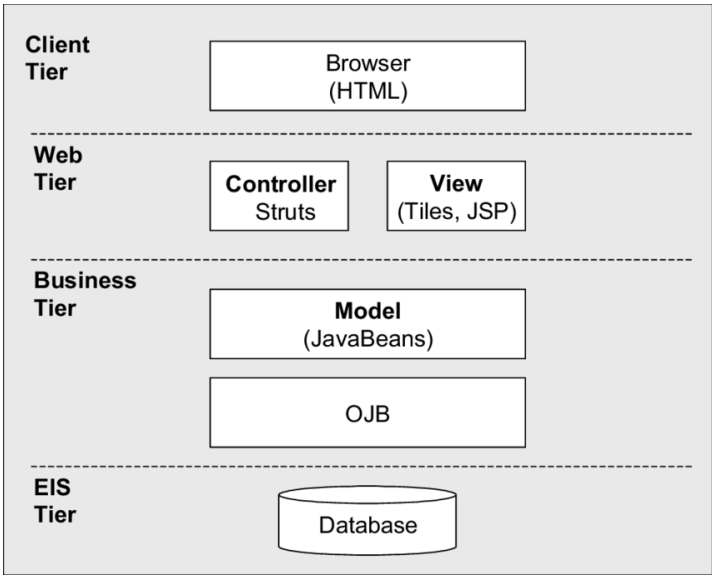


Рисунок 1.9 – Приклад схеми багатошарової архітектури

Загалом, рівні в багаторівневій програмі включають наступне:

- Рівень Виду: надає інтерфейс користувача та доступ до додатків
- Рівень обробки додатків: основна логіка додатка
- Рівень доступу до даних: надає механізм, що використовується для доступу та обробки даних

- Рівень даних: утримує та керує даними.

1.4.6 Веб-сервіси

Веб-сервіси є ключовим моментом інтеграції для різних додатків, що належать до різних платформ, мов, систем.

Веб-сервіси - це набір платформних незалежних API (функцій), які можна використовувати з віддаленого сервера через Інтернет. В основному в цьому беруть участь дві сторони: одна, яка надає набір відкритих API, а друга, зазвичай відома споживачами веб-служб, - це сторона, яка використовує функціональні можливості та послуги, що надаються стороною веб-служб.

Існують різні методи надання веб-служб, але найпоширенішими є SOAP та REST.

Висновки до розділу 1.

Метою даного розділу було оглянути основні типи клієнт-серверних додатків, а також розглянути їхні переваги та недоліки. Достатню увагу було приділено схемі побудови кожного архітектурного рішення, та взаємодії між компонентами.

Все вищенаведене дає змогу зробити вибір на користь тієї чи іншої моделі. Для проекту було вирішено використовувати трьохшарову архітектуру через простоту її реалізації, можливості для розширення та відмовостійкість.

У наступному розділі описано існуючі системи, фреймворки та інструменти для написання власного додатку.

РОЗДІЛ 2. ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ТА ТЕХНОЛОГІЙ

2.1. SOAP

SOAP (Simple Object Access Protocol) - протокол доступу до веб-служб на основі стандартів, який існує вже давно. Спочатку розроблявся корпорацією Майкрософт, SOAP не такий простий, як це може здатись судячи з абревіатури.

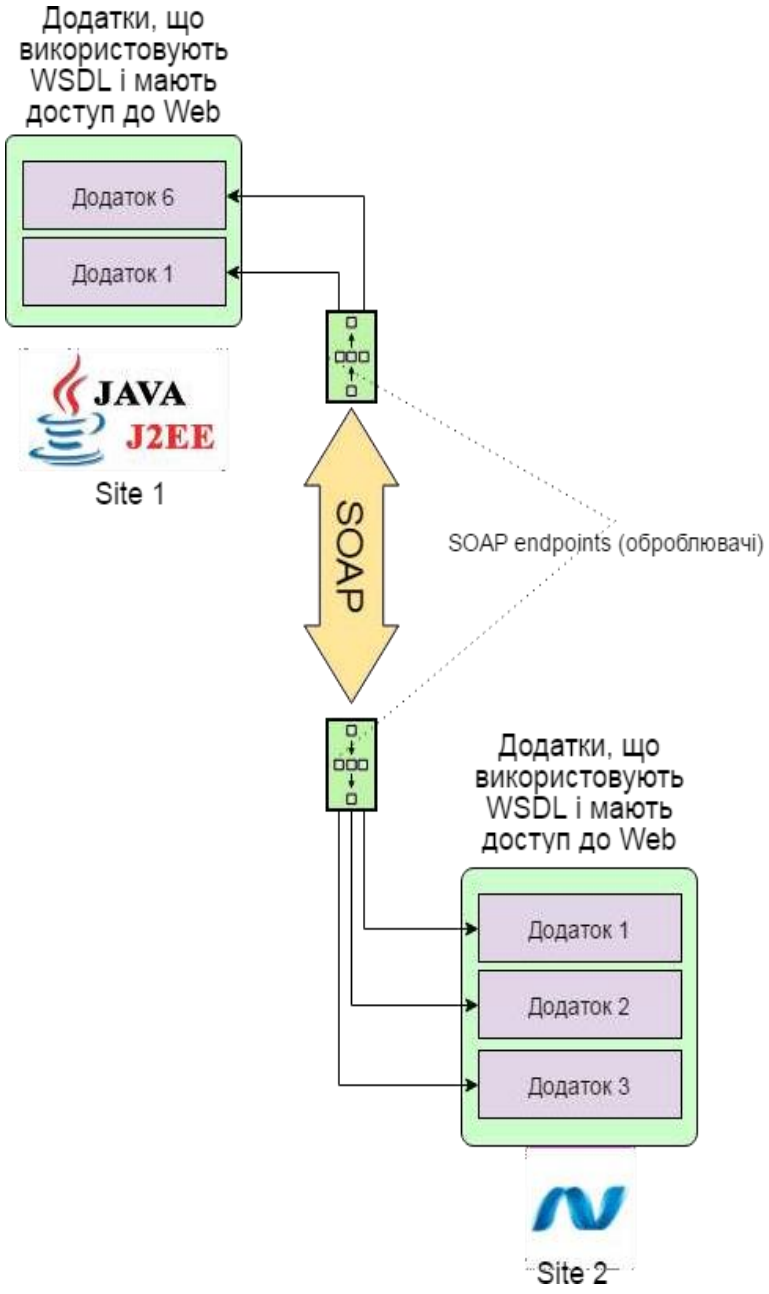


Рисунок 2.1 – Механізм SOAP

SOAP для надання послуг обміну повідомленнями покладається виключно на XML. Microsoft спочатку розробила SOAP, щоб замість нього застосувати

більш старі технології, які не працюють в Інтернеті, такі як модель розподіленого компонентного об'єкта (DCOM) та загальна брокерована архітектура об'єктів (CORBA). Ці технології виходять з ладу, оскільки вони покладаються на двійкові повідомлення. Повідомлення XML, які використовується SOAP, працює краще через Інтернет.

Після первинного випуску Microsoft представила SOAP в Інженерну робочу групу (IETF), де вона була стандартизована. SOAP розроблений для підтримки розширення, тому він містить всілякі інші абревіатури та скорочення, пов'язані з ним, такі як WS-адресація, WS-політика, WS-безпека, WS-Федерація, WS-надійнийMessaging, WS-координація, WS-AtomicTransaction , і WS-RemotePortlets. Насправді, ви можете знайти цілий список цих стандартів у стандартах веб-служб.

Справа в тому, що SOAP може розширюватись, проте ви використовуєте лише ті деталі, які вам потрібні для певного завдання. Наприклад, коли ви користуєтесь загальнодоступною веб-службою, яка є у вільному доступі для всіх, вам не потрібна WS-Security.

2.1.1 Складність залежить від мови програмування

XML, який використовується для отримання запитів та отримання відповідей у SOAP, може стати надзвичайно складним. У деяких мовах програмування потрібно будувати ці запити вручну, що стає проблематичним, оскільки SOAP нетерпимий до помилок. Однак інші мови можуть використовувати ярлики, які надає SOAP. Вони можуть допомогти вам зменшити зусилля, необхідні для створення запиту та для аналізу відповіді. Насправді, працюючи з мовами .NET, ви навіть ніколи не бачите XML.

Частина магії - мова опису веб-служб (WSDL). Це ще один файл, пов’язаний із SOAP. Він надає визначення того, як працює веб-служба, так що, створюючи посилання на неї, IDE може повністю автоматизувати процес. Отже, складність використання SOAP значною мірою залежить від мови, якою ви користуєтесь.

2.1.2 Вбудована обробка помилок

Однією з найважливіших функцій SOAP є вбудована робота з помилками. Якщо у вашому запиті є проблема, відповідь містить інформацію про помилку, яку ви можете використати для усунення проблеми. Зважаючи на те, що ви, можливо, не володієте веб-службою, ця особливість є надзвичайно важливою; інакше вам залишиться гадати, чому все не вийшло. Повідомлення про помилки навіть надає стандартизовані коди, щоб можна було автоматизувати деякі завдання з усунення помилок у вашому коді.

Цікава особливість SOAP полягає в тому, що вам не обов’язково використовувати його для транспорту HTTP. Існує фактична специфікація для використання SOAP через простий протокол передачі пошти (SMTP), і немає ніяких причин, коли ви не можете використовувати його для інших транспортів. Насправді розробники деяких мов, таких як Python та PHP, роблять саме це.

2.2. REST

REST пропонує більш легке рішення. Багато розробників вважають SOAP громіздким і важким у використанні. Наприклад, робота з SOAP в JavaScript означає написання тони коду для виконання простих завдань, оскільки ви повинні створювати необхідну XML-структуру кожного разу.

Замість використання XML для запиту, REST (як правило) покладається на просту URL-адресу. У деяких ситуаціях ви повинні надати додаткову інформацію, але більшість веб-служб, що використовують REST, покладаються виключно на використання підходу URL. REST може використовувати чотири різні дієслова HTTP v1.1 (GET, POST, PUT та DELETE) для виконання завдань.

На відміну від SOAP, REST не потребує використання XML для надання відповіді. Ви можете знайти веб-сервіси на базі REST, які виводять дані в розділених значеннями команд (CSV), нотації об'єкта JavaScript (JSON) та Really Simple Syndication. Суть у тому, що ви можете отримати потрібний результат у формі що легко розбирати, яку ви використовуєте для своєї програми.

Таблиця 2.1 – Відповідність HTTP-функцій SQL-операціям

CRUD-операції	***Ключові слова REST (HTTP)***	***Оператори SQL (база даних)***
CREATE – створити чи додати нові сутності	POST	INSERT
UPDATE – оновити чи редагувати існуючі дані	PUT	UPDATE
READ – зчитати, отримати дані	GET	SELECT
DELETE – видалити інсуючі дані	DELETE	DELETE/DROP

2.3. Рішення між SOAP та REST

Якщо ви не плануєте створити свою власну веб-службу, рішення протоколу, який потрібно використовувати, вже може бути прийнято для вас. Надзвичайно мало веб-сервісів, таких як Amazon, підтримують обидва. У центрі уваги вашого рішення часто зосереджується на тому, який веб-сервіс найкраще відповідає вашим потребам, а не який протокол використовувати.

2.3.1 SOAP проти REST: первинні відмінності

REST працює через одиночний, послідовний інтерфейс для доступу до названих ресурсів. Він найчастіше використовується, коли ви створюєте відкритий API через Інтернет. З іншого боку SOAP, розкриває компоненти логіки додатків як служб, а не даних. Крім того, він працює через різні інтерфейси. Простіше кажучи, REST отримує доступ до даних, поки SOAP виконує операції за допомогою більш стандартизованого набору шаблонів обміну повідомленнями. Однак у більшості випадків або REST, або SOAP можна використовувати для досягнення однакових результатів (і обидва є нескінченно масштабованими), з деякими відмінностями в налаштуванні.

SOAP спочатку був створений корпорацією Майкрософт, і це було значно довше, ніж REST. Це дає перевагу у тому, що він є усталеним протоколом. Але

REST існує вже давно. Крім того, він вийшов на сцену як спосіб отримати доступ до веб-сервісів набагато простішим способом, ніж це можливо для SOAP за допомогою HTTP.

2.3.2 Переваги REST над SOAP

На додаток до використання HTTP для простоти, REST пропонує ряд інших переваг над SOAP:

- REST дозволяє урізноманітнити формати даних, тоді як SOAP дозволяє використовувати лише XML.
- У поєднанні з JSON (який, як правило, краще працює з даними та пропонує швидший аналіз), REST, як правило, вважається простішим.
- Завдяки JSON, REST пропонує кращу підтримку клієнтам браузера.
- REST забезпечує чудову ефективність, особливо завдяки кешуванню інформації, яка не змінена та не динамічна.
- Саме протокол використовується найчастіше для основних сервісів, таких як Yahoo, eBay, Amazon і навіть Google.
- REST зазвичай швидший і використовує меншу пропускну здатність. Також простіше інтегруватися з існуючими веб-сайтами без необхідності переробляти інфраструктуру сайтів. Це дозволяє розробникам працювати швидше, ніж витрачати час на перезапис сайту з нуля. Натомість вони можуть просто додати нову функціональність.
- Проте SOAP залишається кращим протоколом для певних випадків використання. Загальний консенсус серед експертів в наші дні полягає в тому, що REST є типовим кращим протоколом, якщо немає вагомих причин використовувати SOAP (і є деякі випадки, коли SOAP є кращим).

2.3.3 Переваги SOAP над REST

Оскільки ви можете досягти більшості результатів за допомогою будь-якого протоколу, іноді це питання особистої переваги. Однак є деякі випадки використання, які SOAP, як правило, краще підходять. Наприклад, якщо вам потрібна більш міцна безпека, може стати в нагоді підтримка SOAP для WS-Security. Він пропонує деякі додаткові гарантії конфіденційності та цілісності

даних. Він також забезпечує підтримку перевірки ідентичності через посередників, а не просто «точка-точка», як це надає SSL (що підтримується і SOAP, і REST).

Ще одна перевага SOAP полягає в тому, що він пропонує вбудовану логіку для компенсації невдалої комунікації. REST, з іншого боку, не має вбудованої системи обміну повідомленнями. Якщо комунікація не вдається, клієнту доводиться впоратися з цим, повторивши спробу. Також немає стандартного набору правил для REST. Це означає, що обидві сторони (послуга та споживач) повинні розуміти як зміст, так і контекст.

2.3.4 Інші переваги SOAP

Стандартний протокол HTTP SOAP полегшує йому роботу через брандмауері та проксі-сервери без змін самого протоколу SOAP. Але, оскільки він використовує складний формат XML, він, як правило, повільніше порівняно з проміжними програмами, такими як ICE та COBRA.

Крім того, хоча це рідко потрібно, деякі випадки використання вимагають більшої надійності транзакцій, ніж те, що можна досягти HTTP (що обмежує REST в цьому). Якщо вам потрібні транзакції, сумісні з ACID, SOAP - це вихід.

У деяких випадках проектування служб SOAP насправді може бути менш складним порівняно з REST. Для веб-служб, які підтримують складні операції, що вимагають збереження вмісту та контексту, розробка служби SOAP вимагає меншого кодування на рівні додатків для транзакцій, безпеки, довіри та інших елементів.

SOAP може розширюватись за допомогою інших протоколів та технологій. На додаток до WS-Security, SOAP підтримує WS-Addressing, WS-Coordination, WS-ReliableMessaging та безліч інших стандартів веб-служб.

2.4. Огляд Java

Java - це мова програмування та обчислювальна платформа, вперше випущена компанією Sun Microsystems в 1995 році. Є безліч додатків і веб-сайтів, які не працюватимуть, якщо у вас не встановлена Java, і щодня створюється більше. Java є швидкою та надійною. Від ноутбуків до центрів обробки даних,

ігрових консолей до наукових суперкомп'ютерів, мобільних телефонів та Інтернету, Java є скрізь!

Синтаксис Java схожий на C ++, але є строго об'єктно-орієнтованою мовою програмування. Наприклад, більшість програм Java містять класи, які використовуються для визначення об'єктів, та методи, які призначаються окремим класам. Java також відома тим, що є більш строгою, ніж C ++, тобто змінні та функції повинні бути чітко визначені. Це означає, що вихідний код Java може створювати помилки або "exceptions" легше, ніж інші мови, але також обмежує інші типи помилок, які можуть бути викликані невизначеними змінними або непризначеними типами.

На відміну від виконуваних файлів Windows (файли .EXE) або програм Macintosh (файли .APP), програми Java не запускаються безпосередньо операційною системою. Натомість програми Java інтерпретуються віртуальною машиною Java або JVM, яка працює на декількох платформах. Це означає, що всі програми Java є багатоплатформенними та можуть працювати на різних платформах, включаючи комп'ютери Macintosh, Windows та Unix. Однак JVM повинен бути встановлений, щоб програми Java або аплети взагалі працювали. На щастя, JVM включений як частина Java Runtime Environment (JRE)

Остання версія Java містить важливі покращення для підвищення продуктивності, стабільності та безпеки додатків Java, які працюють на вашому пристрої. Встановлення цього безкоштовного оновлення забезпечить безпечну та ефективну роботу програм Java..

2.5. Огляд Spring Framework

Spring Framework (Spring) - це програма з відкритим кодом, що забезпечує інфраструктурну підтримку для розробки програм Java. Один з найпопулярніших фреймворків Java Enterprise Edition (Java EE), Spring допомагає розробникам створювати високоефективні програми, використовуючи звичайні старі об’єкти Java (POJO).

Фреймворк - це великий масив заздалегідь заданого коду, до якого розробники можуть додавати код для вирішення проблеми в певній області. Існує

багато популярних фреймворків Java, включаючи Java Server Faces (JSF), Maven, Hibernate, Struts та Spring.

Програми Java складні і містять багато важких компонентів. Велика вага означає, що компоненти залежать від операційної системи (ОС) за своїм зовнішнім виглядом та властивостями.

Spring вважається надійним, недорогим та гнучким фреймворком. Spring підвищує ефективність кодування та скорочує загальний час розробки додатків, оскільки він легкий (ефективний при використанні системних ресурсів) і має велику підтримку.

Spring знімає складну роботу над конфігураціями, щоб розробники могли зосередитись на написанні ділової логіки. Spring обробляє інфраструктуру, щоб розробники могли зосередити увагу на додатку.

Spring Framework поділений на модулі. Програми можуть вибирати, які модулі їм потрібні. В основі лежать модулі основного контейнера, включаючи модель конфігурації та механізм введення залежності. Крім цього, Spring Framework надає основоположну підтримку різних архітектур прикладних програм, включаючи обмін повідомленнями, транзакційні дані та постійність та Інтернет. Він також включає серверну веб-структуру Spring MVC на основі сервлетів і, паралельно, реактивну веб-структуру Spring WebFlux.

Коли ви дізнаєтесь про фреймворки, важливо знати не тільки те, що він робить, але і яких принципів він дотримується. Ось керівні принципи Spring Framework:

- Забезпечити вибір на кожному рівні. Spring дозволяє відкладати дизайнерські рішення якомога пізніше. Наприклад, ви можете перемикати реалізації через конфігурацію, не змінюючи код. Те саме стосується багатьох інших інфраструктурних проблем та інтеграції з сторонніми API.
- Визначити різноманітні точки зору. Spring гнучкий. Він підтримує широкий спектр потреб у застосуванні з різними перспективами.
- Підтримувати зворотну сумісність. Spring підтримує ретельно підібраний діапазон версій JDK та сторонніх бібліотек для полегшення обслуговування

додатків і бібліотек, які залежать від Spring.

• Дбати про дизайн API. Команда Spring приділяє багато думок та часу для створення інтуїтивно зрозумілих API, які підтримують багато версій протягом багатьох років.

• Встановити високі стандарти якості коду. Spring Framework робить великий акцент на змістовній, актуальній та точній документації. Це один з дуже небагатьох проектів, який може зберегти чисту структуру коду без кругових залежностей між пакетами.

2.6. Огляд MongoDB

MongoDB - орієнтована на документи база даних NoSQL, яка використовується для зберігання даних з великим обсягом. Замість використання таблиць та рядків, як у традиційних реляційних базах даних, MongoDB використовує колекції та документи. Документи складаються з пар ключових значень, які є базовою одиницею даних у MongoDB. Колекції містять набори документів та функції, що є еквівалентом таблиць реляційних баз даних. MongoDB - це база даних, яка з'явилася на світ приблизно в середині 2000-х.

Особливості MongoDB:

• Кожна база даних містить колекції, які в свою чергу містять документи. Кожен документ може бути різним із різною кількістю полів. Розмір і зміст кожного документа можуть відрізнятися один від одного.

• Структура документа більше відповідає тому, як розробники конструюють свої класи та об’єкти у відповідних мовах програмування. Розробники часто скажуть, що їхні класи не є рядками та стовпцями, але мають чітку структуру з парами ключ-значення.

• У рядках (або документах, як їх називають у MongoDB) не потрібно заздалегідь мати схему. Натомість поля можна створювати на льоту.

• Модель даних, доступна в MongoDB, дозволяє представити ієрархічні відносини, легше зберігати масиви та інші більш складні структури.

2.7. Огляд MySql

Бази даних MySQL є реляційними.

Реляційна база даних зберігає дані в окремих таблицях, а не зберігає всі дані в одному великому сховищі. Структури бази даних організовані у фізичні файли, оптимізовані для швидкості. Логічна модель з такими об'єктами, як бази даних, таблиці, подання, рядки та стовпці пропонує гнучке середовище програмування. Ви встановлюєте правила, що регулюють взаємозв'язки між різними полями даних, наприклад, один на один, один на багато, унікальний, необхідний або необов'язковий та "покажчики" між різними таблицями. База даних виконує ці правила, так що при добре розробленій базі даних ваша програма ніколи не бачить непослідовних, дублікатів, сиріт, застарілих або відсутніх даних.

SQL частина "MySQL" означає "Структурована мова запитів". SQL - найпоширеніша стандартизована мова, що використовується для доступу до баз даних. Залежно від середовища програмування, ви можете вводити SQL безпосередньо (наприклад, для генерації звітів), вставляти SQL заяви в код, написаний іншою мовою, або використовувати специфічний для мови API, який приховує синтаксис SQL[11, ст 49-52].

SQL визначається стандартом ANSI / ISO SQL. Стандарт SQL розвивається з 1986 року і існує кілька версій. У цьому посібнику "SQL-92" посилається на стандарт, випущений у 1992 році, "SQL: 1999" відноситься до стандарту, випущеного в 1999 році, а "SQL: 2003" стосується поточної версії стандарту. Ми використовуємо фразу "стандарт SQL", щоб означати поточну версію стандарту SQL в будь-який час. [11, ст 52-55]

Програмне забезпечення баз даних MySQL - це система клієнт / сервер, що складається з багатопотокового SQL-сервера, який підтримує різні тижні, декілька різних клієнтських програм і бібліотек, адміністративні засоби та широкий спектр інтерфейсів прикладного програмування (API).

Доступна велика кількість програмного забезпечення MySQL.

MySQL Server має практичний набір функцій, розроблений у тісній співпраці з нашими користувачами. Ймовірно, що ваша улюблена програма чи мова підтримує сервер баз даних MySQL[11, ст 55-59]

2.8. Огляд Thymeleaf

Thymeleaf - це сучасний серверний механізм Java для веб сторінок.

Основна мета Thymeleaf - привнести елегантні природні шаблони у ваш робочий процес розвитку - HTML, який можна правильно відображати в браузерах, а також працювати зі статичними прототипами, що дозволяє посилити співпрацю в командах розвитку.

З модулями для Spring Framework, безліччю інтеграцій з улюбленими інструментами та можливістю підключення власного функціоналу Thymeleaf ідеально підходить для сучасної веб-розробки HTML5 JVM в Інтернеті, хоча він може набагато більше.

Thymeleaf - надзвичайно розширюваний механізм (насправді його можна назвати платформою шаблонів), який дозволяє вам визначати і налаштовувати спосіб обробки ваших шаблонів до тонкого рівня деталізації.

Об'єкт, який застосовує деяку логіку до артефакту розмітки (тегу, текст, коментарю) називається процесором, а набір цих процесорів - плюс, можливо, деякі додаткові артефакти - це те, з чого складається діалект. З коробки основна бібліотека Thymeleaf надає діалект, званий стандартним діалектом, якого повинно бути достатньо для більшості користувачів.

2.9. Огляд існуючих програмних рішень

Існуючі програмні рішення зазвичай пропонують однаковий користувацький функціонал:

- Поточний розклад кіносеансів(Рисунки 2.1, 2.2, 2.3)
- Майбутні кінопрем’єри (Рисунки 2.4, 2.5, 2.6)
- Інформаційні сторінки (Рисунки 2.7, 2.8, 2.9)
- Особистий кабінет (Рисунки 2.10, 2.11, 2.12)

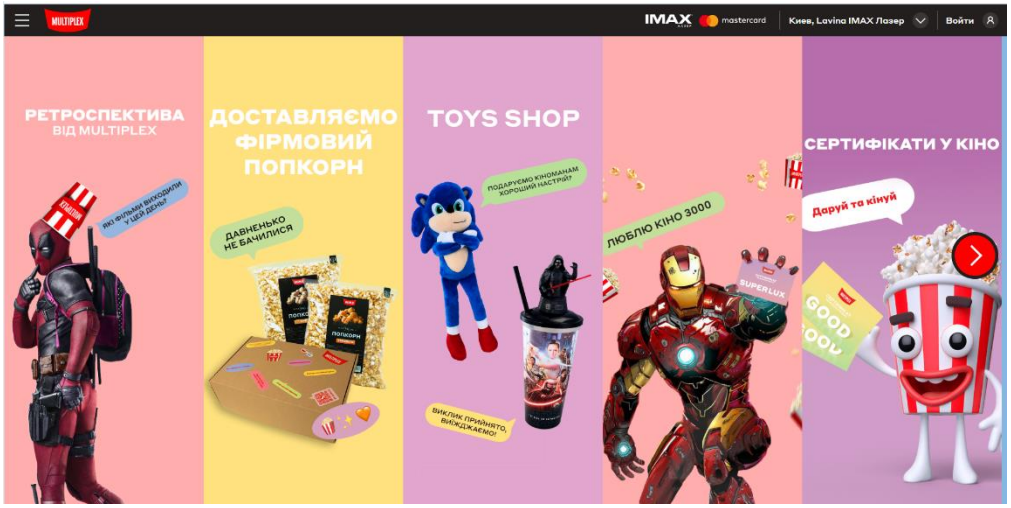


Рисунок 2.1 – Поточний розклад кінотеатру Multiplex

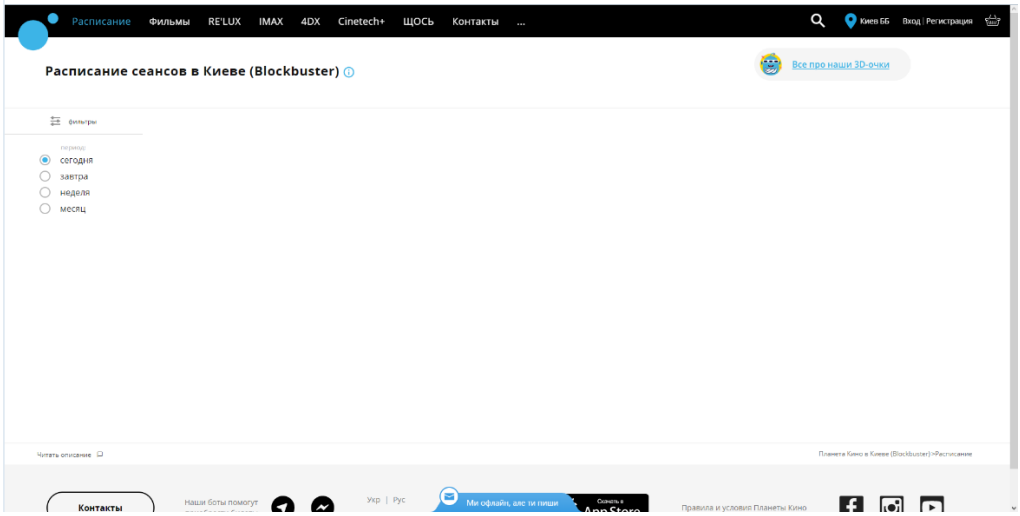


Рисунок 2.2 – Поточний розклад кінотеатру Планета кіно

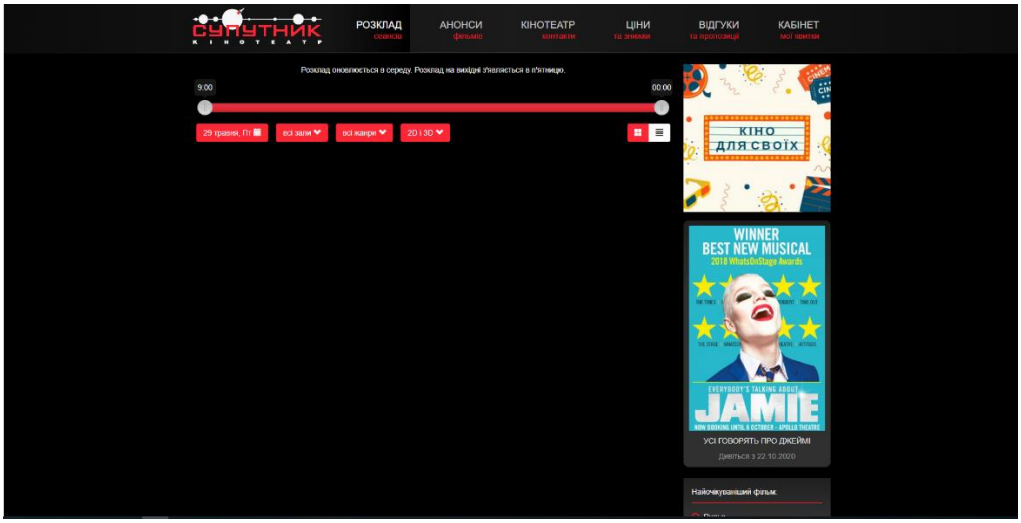


Рисунок 2.3 – Поточний розклад кінотеатру Супутник

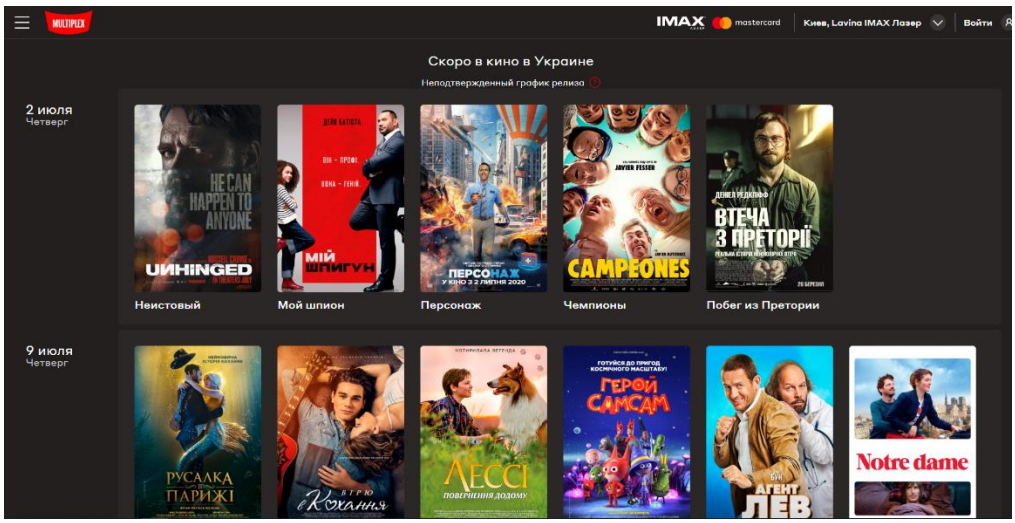


Рисунок 2.4 – Прем’єри кінотеатр Multiplex

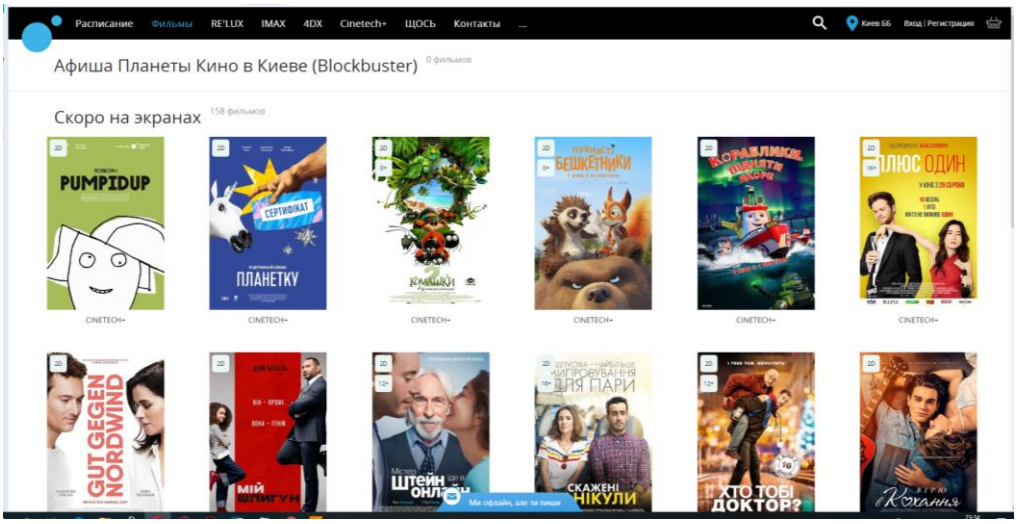


Рисунок 2.5 – Прем’єри кінотеатр Планета кіно

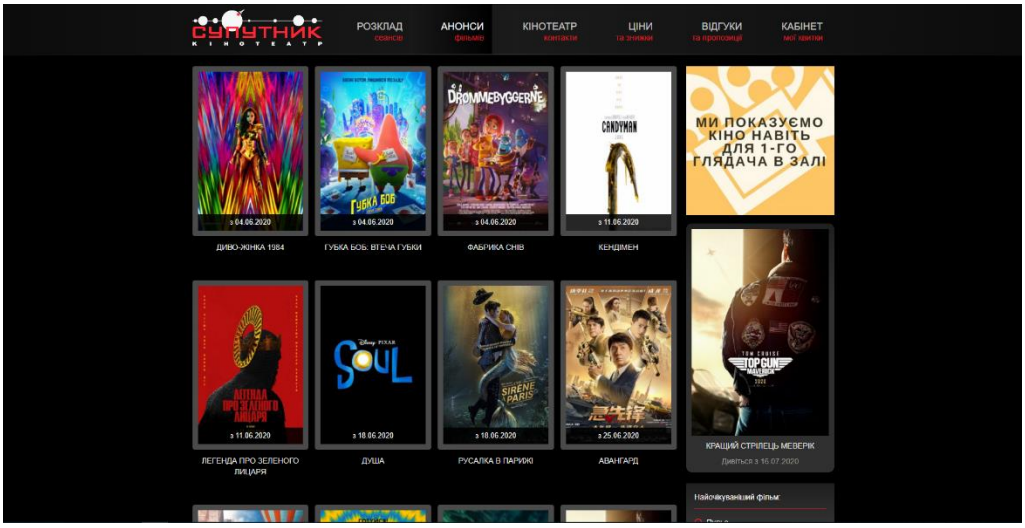


Рисунок 2.6 – Прем’єри кінотеатр Супутник

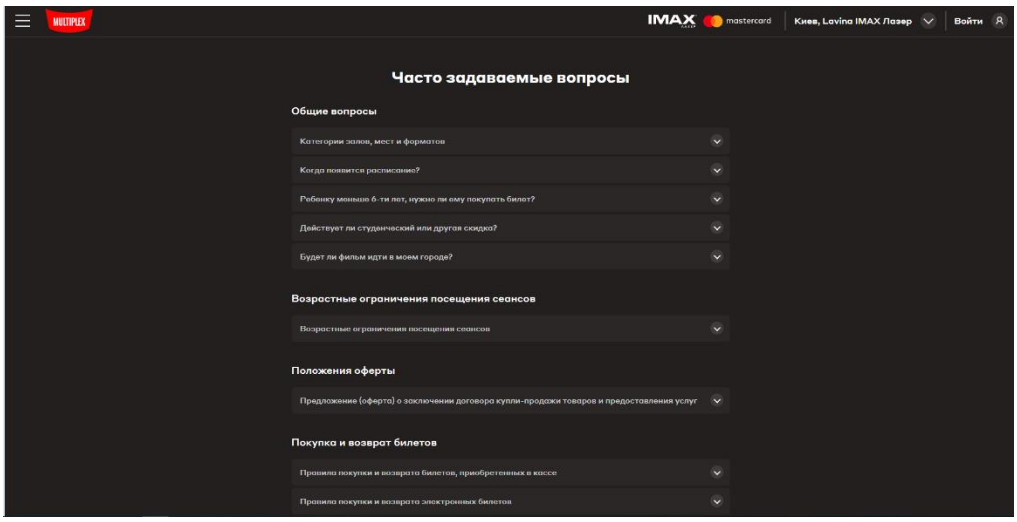


Рисунок 2.7 – Інформаційна сторінка кінотеатру Multiplex

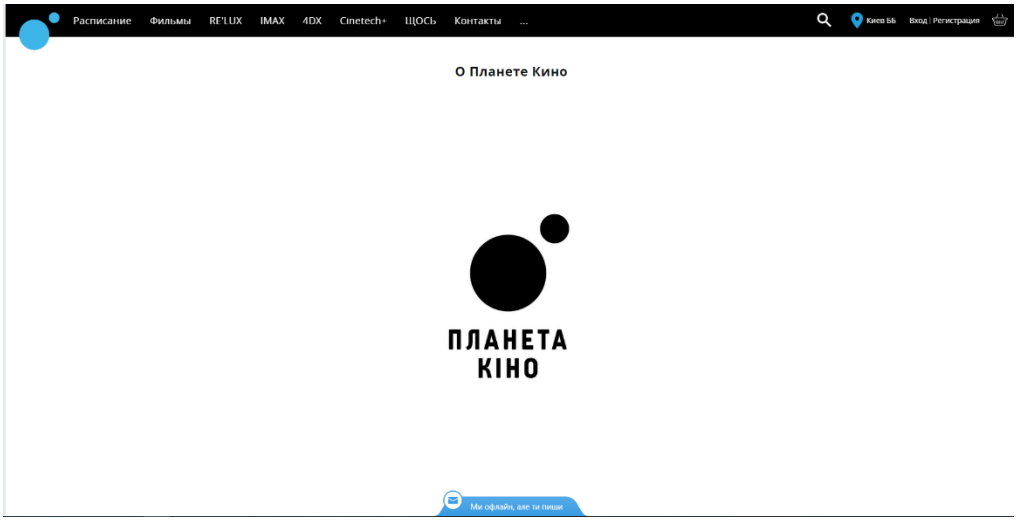


Рисунок 2.8 – Інформаційна сторінка кінотеатру Планета кіно

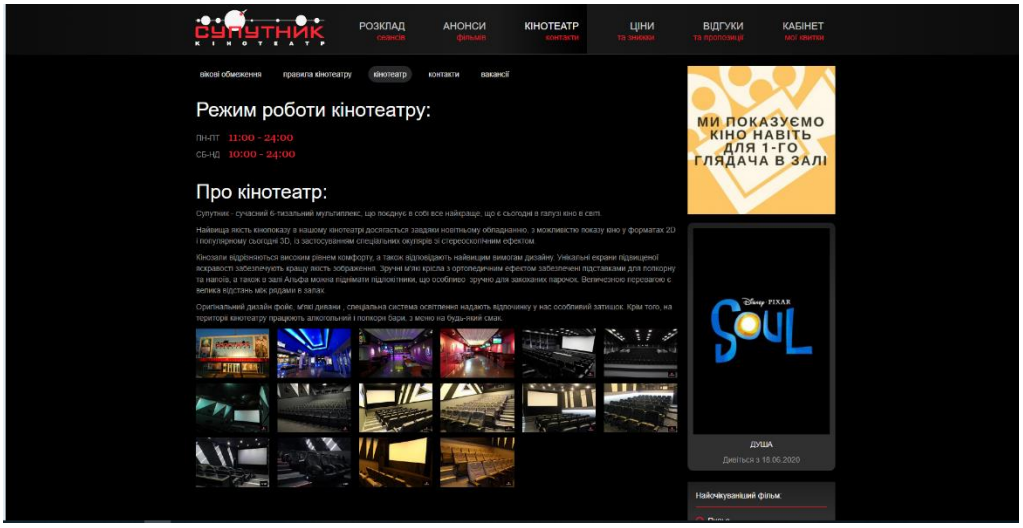


Рисунок 2.9 – Інформаційна сторінка кінотеатру Супутник

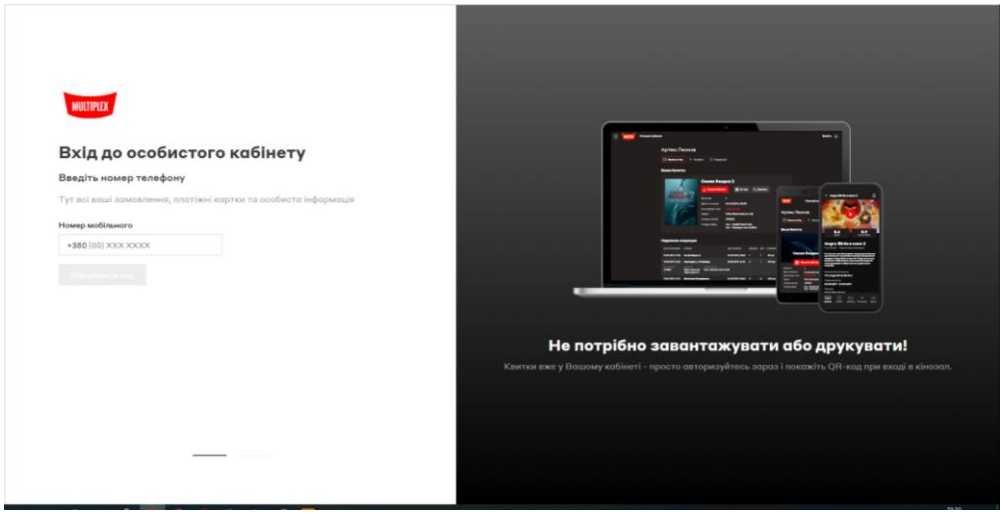


Рисунок 2.10 – Вхід до особистого кабінету кінотеатр Multiplex

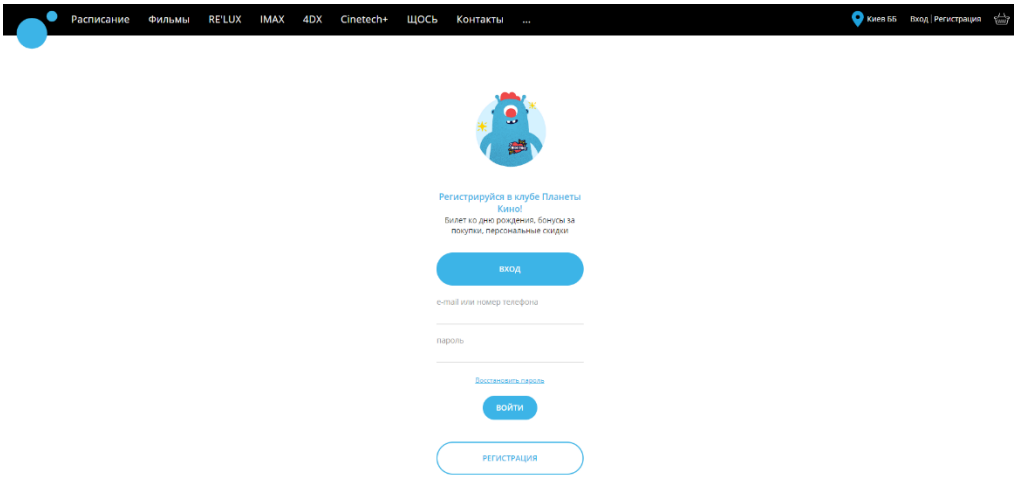


Рисунок 2.11 – Вхід до особистого кабінету кінотеатр Планета кіно

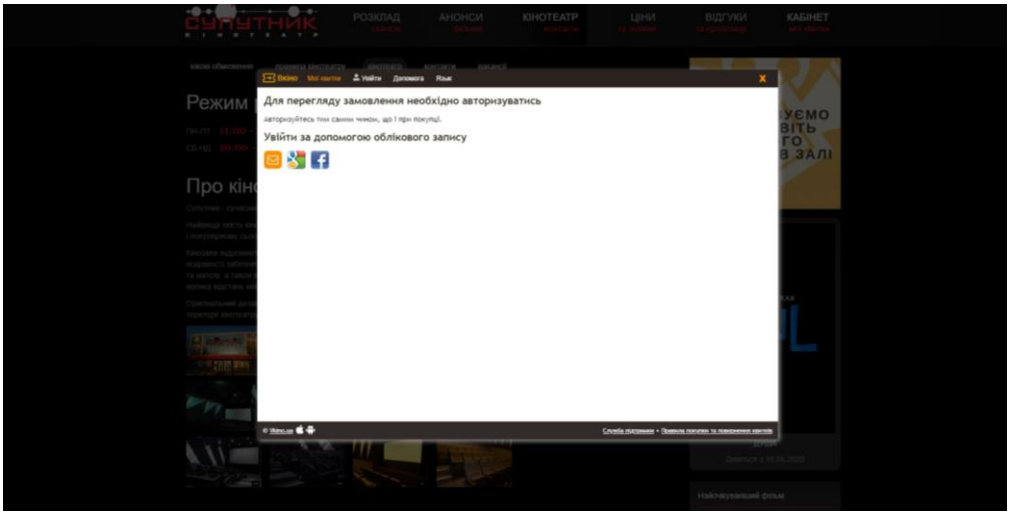


Рисунок 2.12 – Вхід до особистого кабінету кінотеатр Супутник

Висновки до розділу 2.

Для створення власного веб-сервера, було вирішено використати Java та Spring Framework так як саме ці технології дозволяють створити оптимальне рішення, що буде задовольняти клієнтські потреби. .

Для роботи з базами даних було вирішено обрати MySql через її зручність у зберіганні даних, та її релятивність. Також ця база даних має ряд програмних інтерфейсів, які дозволяють встановлювати з’єднання з сервером написаним за допомогою Spring Framework.

Також було вирішено реалізовувати RESTful архітектуру, це обумовлено технічними потребами системи, невеликим навантаженням на неї та простоту підтримки додатку.

В кінці даного розділу було окреслено основні функціональні особливості клієнт-серверних застосунків для керування системами кінотеатрів.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ

Для реалізації даного клієнт-серверного застосунку було вирішено обрати архітектуру з клієнтом-терміналом та «Товстим сервером».

Основні компоненти будуть відповідати класичній схемі Модель-Вид-Контролер

Модель являє собою об’єктно-реляційне представлення бази даних, Spring репозиторії та сервіси для виконання основної бізнес логіки

Контролером виступає стандартний контролер з фреймворку Spring. Він обробляє всі запити що надсилаються на сервер.

Для реалізації Виду було використано шаблоннізатор Thymeleaf, HTML5, CSS та JS.

3.1 Структура бази даних

Для додатку була спроектована база даних(Рисунок 3.1).

Структура бази обумовлена даними які будуть зберігатися в ній.

Короткий опис таблиць:

- Movie_theatre – таблиця для зберігання інформації про кінотеатр
- Hall – тут зберігається інформація про всі зали кінотеатру
- Session – дані про сесії кінофільмів
- Place – інформація про місця в кінотеатрі
- Ticket – тут зберігаються всі квитки куплені користувачами
- User – інформація про клієнтів кінотеатру
- Movie – всі фільми що були показані в кінотеатрі а також майбутні прем’єри
- Trailer – трейлери для фільмів
- Movie_genre – жанри фільмів
- Movie_director – режисери фільмів

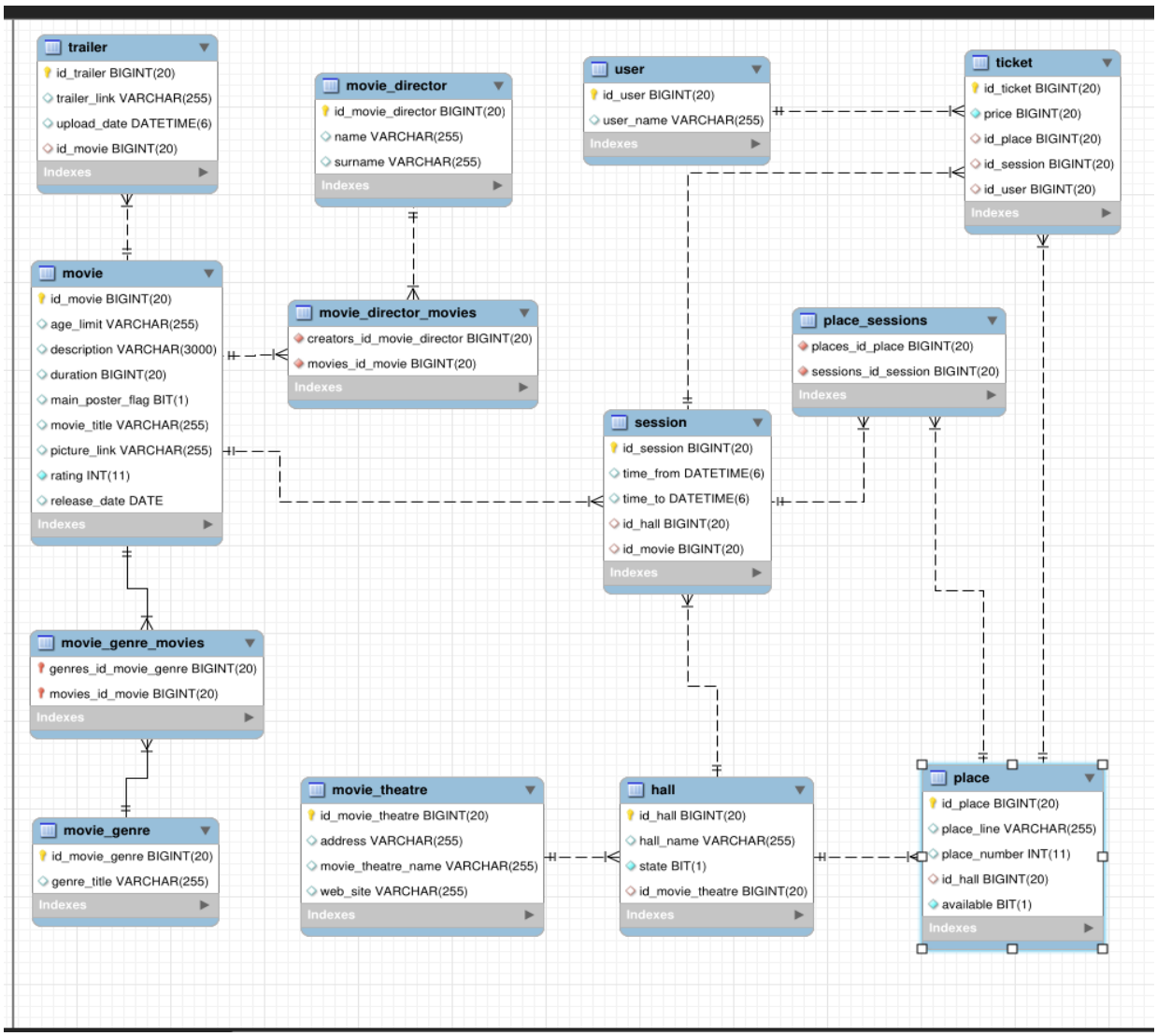


Рисунок 3.1 – Структура бази даних

Також було створено додаткові таблиці для реалізації відношень Багато-багато:

- Movie_genre_movies – відношення фільмів та їхніх жанрів
- Movie_director_movies – відношення фільмів та режисерів
- Place_sessions – відношення кіносеансів та місць

3.2 Структура проекту

Для огранізації коректної роботи потрібно створити правильну структуру проекту. Зазвичай структура Spring проекту включає стандартні директорії :

Config – директорія в якій розміщені конфігураційні файли

Model – директорія в якій знаходяться головні компоненти роботи з базою даних(Entity, Data Transfer Objects та репозиторії для бази)

Service – директорія в якій зберігаються класи що відповідають за бізнес логіку

Controller – директорія в якій містяться контролери що обробляють HTTP запити

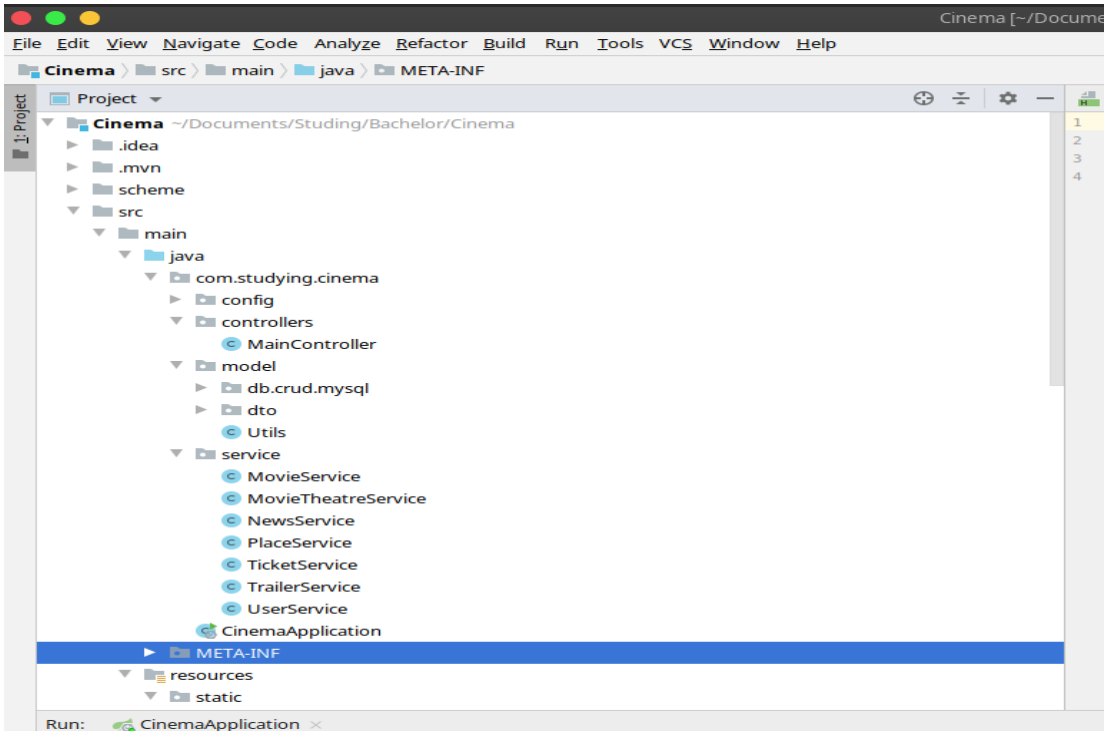


Рисунок 3.2 – Структура проекту

3.3 Структура відображення графічного інтерфейсу користувача

Графічний інтерфейс користувача одна з найголовніших частин проекту, адже саме цю частину буде бачити кінцевий користувач. З цього можна зробити висновок що проектування front-end частини є дуже важливим етапом створення додатку. На Рисунку 3.3 наведено структуру програми яка описує логіку вигляду.

Зовнішній вигляд готового продукту надзвичайно важливий пункт, який варто враховувати при розробці програмного забезпечення. Отож було вирішено використовувати Bootstrap та JS щоб зробити інтерфейс більш дружнім для користувача та зробити його більш привабливим. На рисунках 3.4, 3.5, 3.6 наведено приклад зовнішнього вигляду розробленого продукту.

Рисунок 3.3 – Структура front-end частини

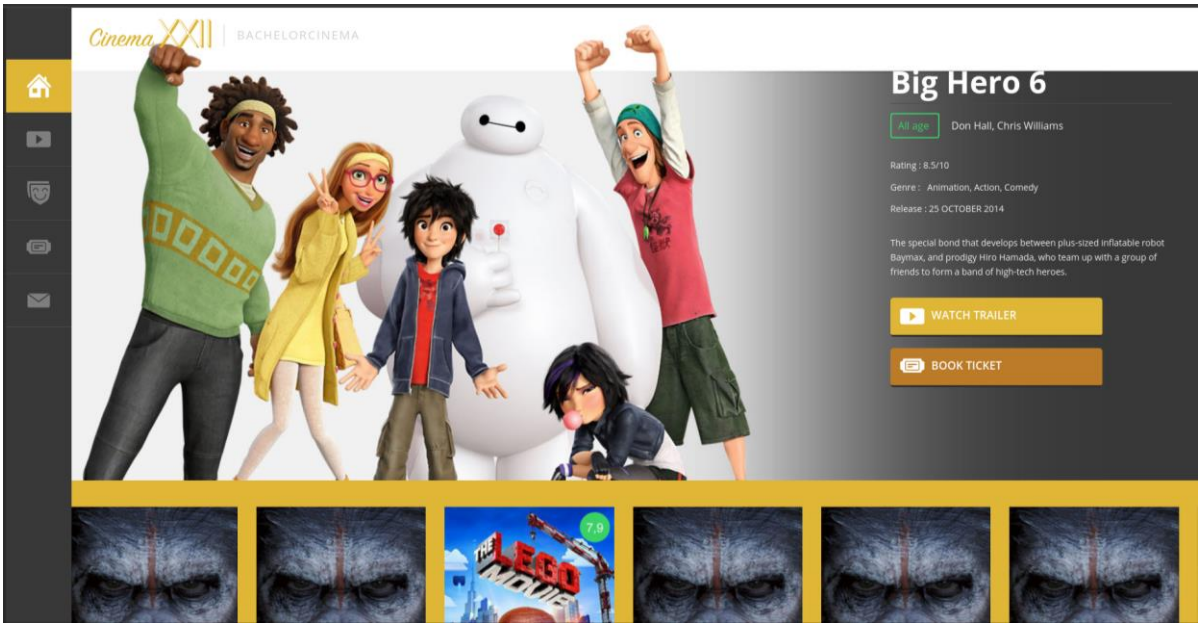


Рисунок 3.3 – Дизайн головної сторінки проекту (частина 1)

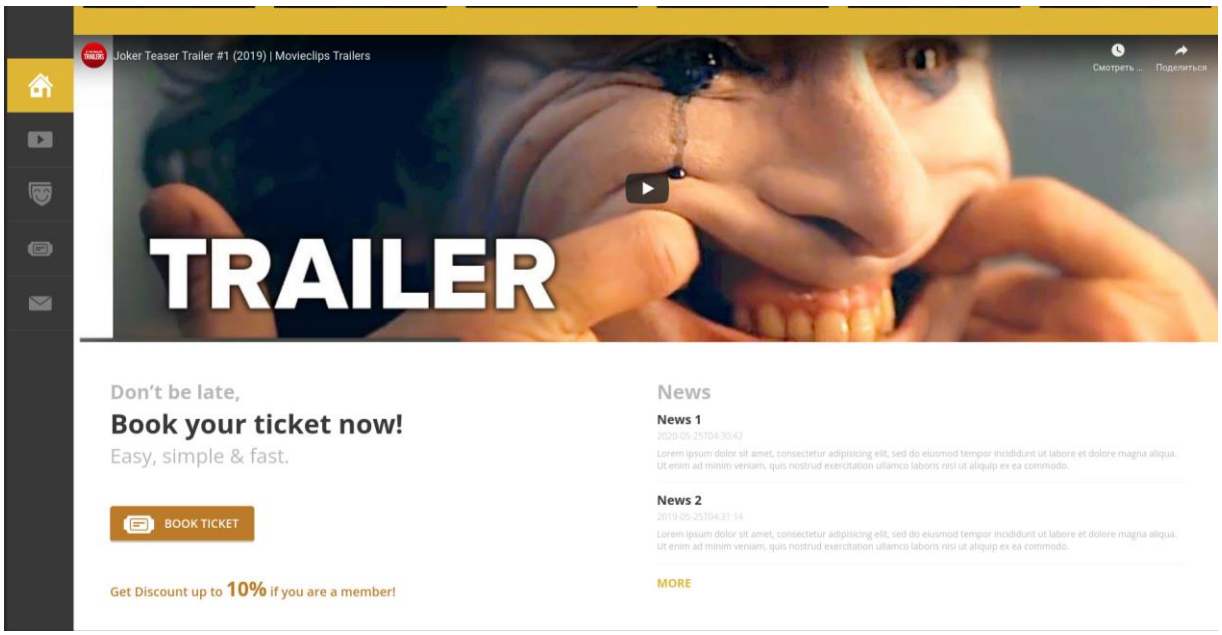


Рисунок 3.4 – Дизайн головної сторінки проекту (частина 2)

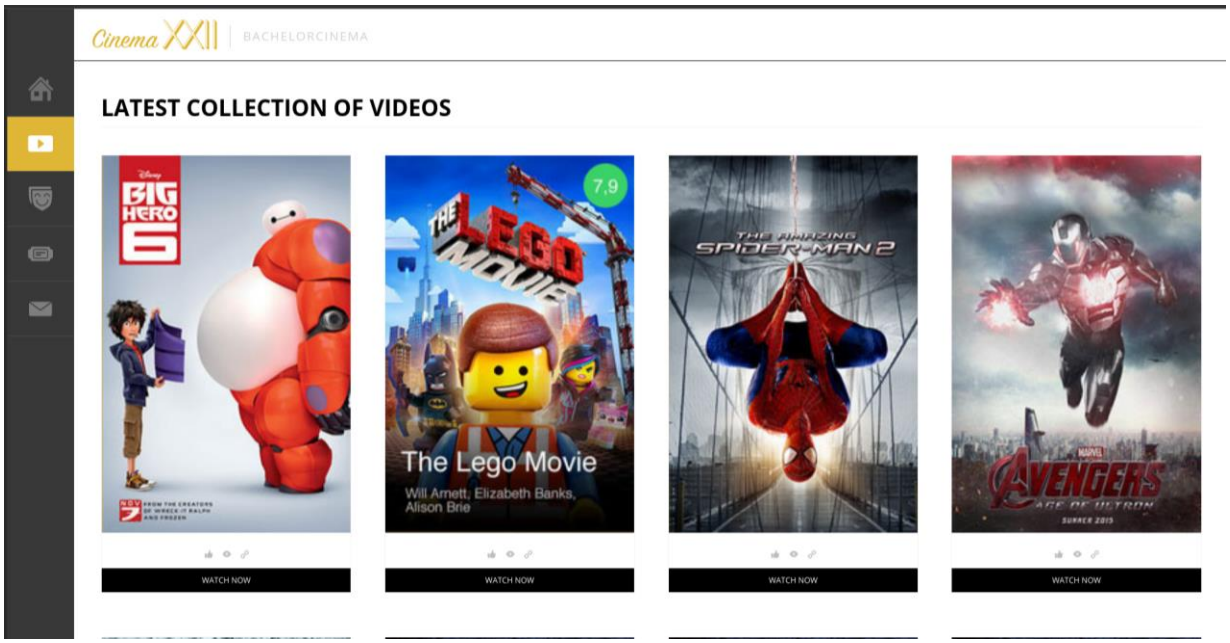


Рисунок 3.5 – Дизайн сторінки з оглядом всіх фільмів в кінотеатрі

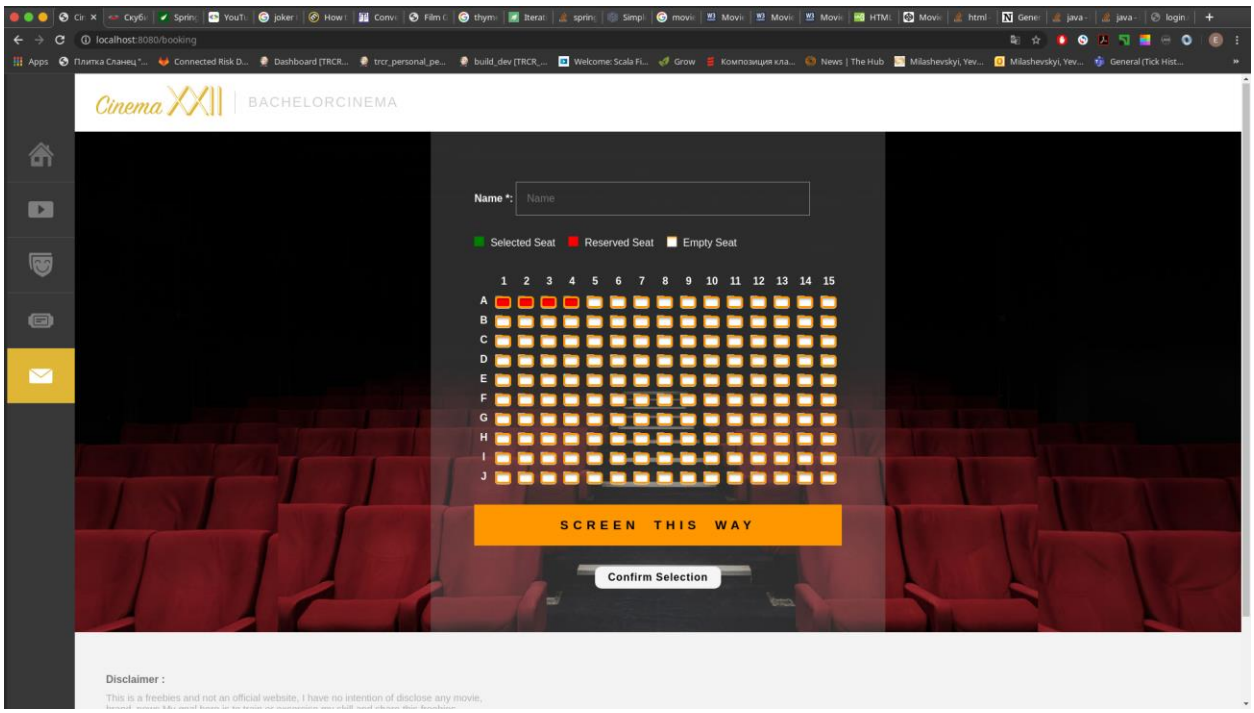


Рисунок 3.6 – Дизайн сторінки для бронювання квитків

3.4 Тестування продукту

Тестування продукту на етапі розробки, а також тестування готового рішення це невід'ємна частина роботи. Добре протестований та покрити юніт тестами додаток підвищує свою відмовостійкість, полегшує процес подальшої розробки та майбутньої підтримки проекту. На рисунку 3.7 наведено схему що включає тести.

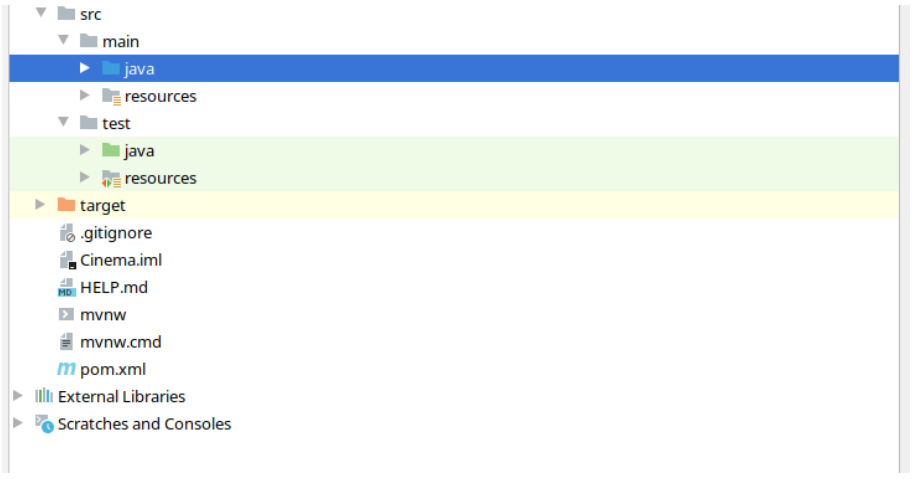


Рисунок 3.7 – Коренева структура проекту

3.5 Огляд основних компонентних класів додатку

Класи з пакету com.studying.cinema.model.db.crud.mysql.entity – основні

класи для роботи з базою даних. Об’єкти цих класів являють собою прості Java об’єкти що мають тільки стан та не мають поведінки.

Класи з пакету com.studying.cinema.model.db.crud.mysql.repositories – класи що допомагають виконувати запити до бази даних та перетворювати табличні значення у прості Java об’єкти.

Класи з пакету com.studying.cinema.model.dto – класи для транспортування даних від бази даних до шару представлення. Створення таких класів вважається хорошою практикою при проектуванні клієнт-серверних додатків.

Клас MainController це ключовий компонент створеного додатку. Саме цей клас виконує обробку всіх запитів що відправляються користувачем. Головний контролер реалізований стандартними засобами Spring фреймворку.

Клас CinemaApplication є точкою запуску додатку, самец ей клас розгортає всю інфраструктуру необхідну для роботи додатку.

Класи-сервіси з пакету com.studying.cinema.service регулють всю роботу бізнес шару. Саме ці класи викликають методи для роботи з базою даних, опрацьовують дані, валідують, сортують, перетворюють на транспортну об’єкти. На рисунку 3.8 зображено всі сервісні класи.

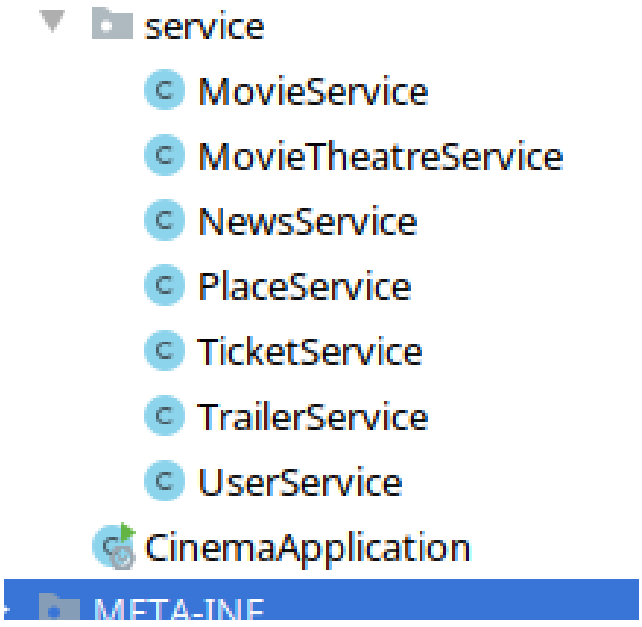


Рисунок 3.8 – Класи сервіси

Кожен з класів-сервісів відповідає за свою частину бізнес логіки. Таким чином MovieTheatreService виконую всі функції пов’язані з обробкою даних

кінотеатру, а клас MovieService виконує головну логіку оброки всіх запитів пов’язаних з кінофільмами(дістати з бази даних, відсортувати, відфільтрувати, знайти найновіший трейлер, знайти сеанси на яких буде показано дану кінострічку).

NewsService займається пошуком актуальних новин та створенням нових.

PlaceService та TicketService – сервісні класи, що займаються створенням квитків, пошуком вільних місць та резервуванням вибраних місць.

TrailerService повертає всі трейлери відповідно до фільму, а також найновіший трейлер, відповідно до запиту користувача.

Висновок до розділу 3

Даний розділ описує головні моменти у створенні клієнт-серверного додатку для керування системою кінотеатру. Для написання програмного коду для цього продукту було використано мову Java, фреймворки Spring та Thymeleaf. Для зберігання даних було використано базу даних під керуванням MySql. Бізнес-логіка яку потрібно було реалізувати не є примітивною, саме тому є ще досить багато проектних рішень які можна реалізувати

ВИСНОВКИ

Метою данної дипломної роботи було оглянути типи клієнт-серверних архітектур. Після проведення дослідження, огляду всіх переваг та недоліків різних типів архітектур було зроблено вибір на користь клієнт-серверного додатку на базі серверу.

У першому розділі були описані причини через які тришарову архітектуру вибирають більшість розробників. У наступному розділі окреслено найбільш популярні способи взаємодії між клієнтом та сервером – REST та SOAP, а також найпопулярніші інструменти для реалізації цієї взаємодії. В кінці другого розділу було наведено приклади існуючих програмних рішень. На основі цих даних було вирішено використовувати тришарову архітектуру з «тонким» клієнтом та RESTful взаємодією між компонентами.

Огляд існуючих додатків для керування системами кінотеатрів окреслив основні функціональні вимоги для розроблюваного застосунку.

Розгляд інструментів для реалізації серверної частини нашого рішення показав що саме ці рішення будуть оптимальними для створення конкурентоспроможного продукту.

Під час реалізації було спроектовано схему бази даних та структурну схему проекту. Вибрані інструменти допомогли найкращим чином реалізувати поставлену мету. Готове рішення було ретельно протестоване. Це означає що систему можна буде з легкістю оптимізувати та додати нову функціональність.

Після виконання даної роботи можна зробити висновок що використані інструменти та спроектована архітектура повною мірою покрили всі сценарії роботи додатку, а їх використання було найбільш економічно вигідним.

Також варто зазначити що створена система має повністю відкрита для розширення та впровадження нових функцій, саме це дозволить в подальшому масштабувати систему.

Список використаної літератури

[1] Buschmann, Frank; Meunier, Regine; Rohnert, Hans; Sommerlad, Peter; Stal, Michael (1996-08). Pattern-Oriented Software Architecture, Volume 1, A System of Patterns. Wiley, August 1996. ISBN 978-0-471-95869-7. Retrieved from http://www.wiley.com/WileyCDA/WileyTitle/productCd-0471958697.html

[2] Deployment Patterns (Microsoft Enterprise Architecture, Patterns, and Practices)

[3] Kruchten, Philippe. The Rational Unified Process-An Introduction, Addison-Wesley, 1998

[4] Mak, Gary (September 1, 2010). Spring Recipes: A Problem-Solution Approach (Second ed.). Apress. p. 1104. ISBN 1-4302-2499-1.

[5] Rumbaugh, J., Jacobsen, I. and Booch, G. The Unified Modelling Language Reference Manual. Reading, Mass.: Addison-Wesley, 1999

[6] "SQL". Britannica.com. Retrieved 2013-04-02. https://www.britannica.com/technology/SQL

[7] Thymeleaf 3.0.11 https://www.thymeleaf.org/ Release announcement

[8] "Web Services Architecture". World Wide Web Consortium. 11 February 2004. 3.1.3 Relationship to the World Wide Web and REST Architectures. Retrieved 29 September 2016.

[9] "Web Services Addressing (WS-Addressing)". www.w3.org. Retrieved 2016-09-15.

[10] What is MySQL? MySQL 5.7 Reference Manual. http://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html 1.3.1

[11] Віктор Руденко, Наталія Речич, Валентина Потієнко «Інформатика. Профільний рівень» 2019

[12] Електронний ресурс - https://hub.packtpub.com/what-is-multi-layered-software-architecture/

Додатки

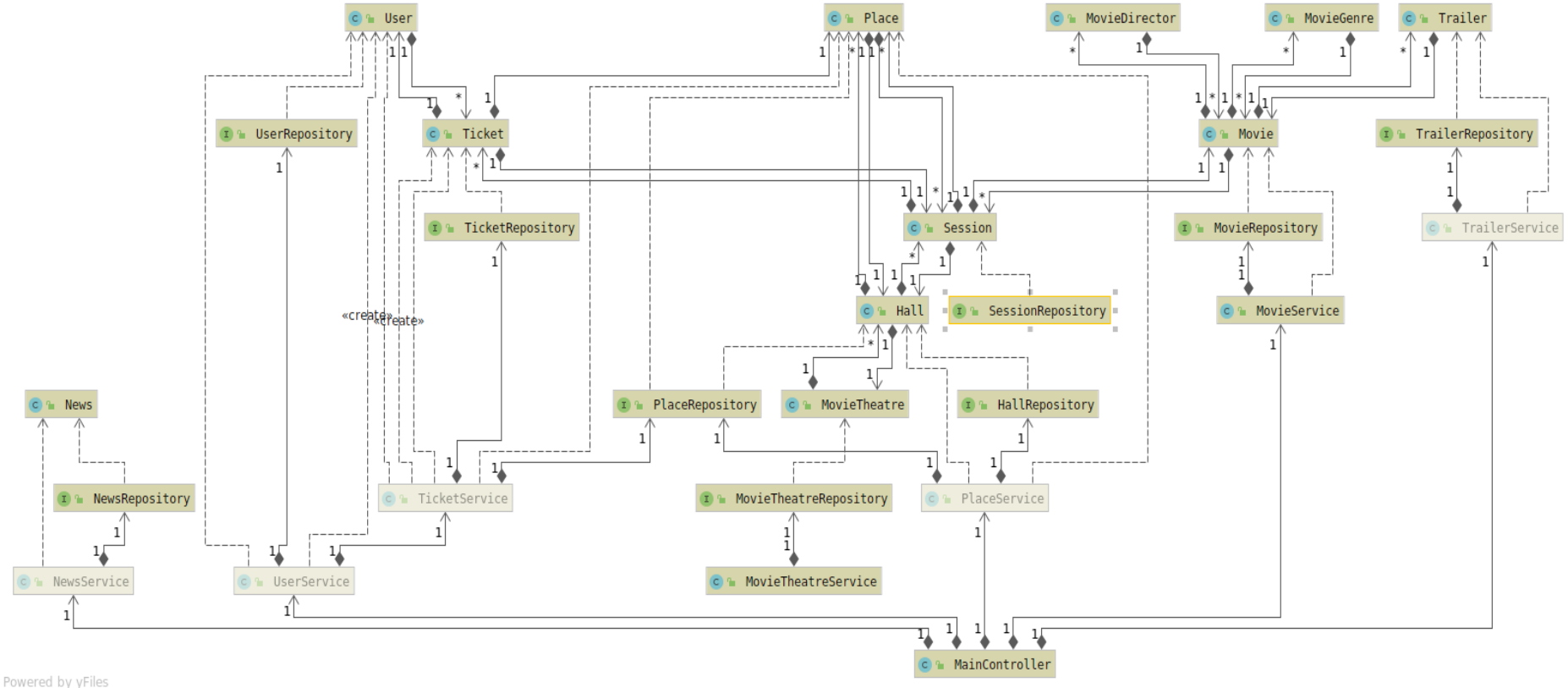
ДОДАТОК А

Клієнт-серверний додаток для керування системою кінотеатру

Функціональна схема

ІАЛЦ.467400.003 Д1

Аркушів 1



					ІАЛЦ 467400 Д1		
Зм.	Арк.	№ докум.	Підпис	Дата	Схема функціональна Діаграма класів	Літ.	Маса / Масштаб
Розроб.		Мілашевський Є					
Перевір.		Ружевський М.С.				Арк.	Аркушів
Н. контр.		Симоненко В.П.			*Дипломна робота*	КПІ ФІОТ кафедра ОТ гр. ІП-62	
Затверд.							

ДОДАТОК Б

Клієнт-серверний додаток для керування системою кінотеатру

Структурна схема
ІАЛЦ.467400.004 Д2

Аркушів 1

					ІАЛЦ 467400 Д2		
Зм.	Арк.	№ докум.	Підпис	Дата	Схема структурна Структура бази даних	Літ.	Маса / Масштаб
Розроб.		Мілашевський Є					
Перевір.		Ружевський М.С.				Арк.	Аркушів
Н. контр.		Симоненко В.П.			*Дипломна робота*	КПІ ФІОТ кафедра ОТ гр. ІП-62	
Затверд.							

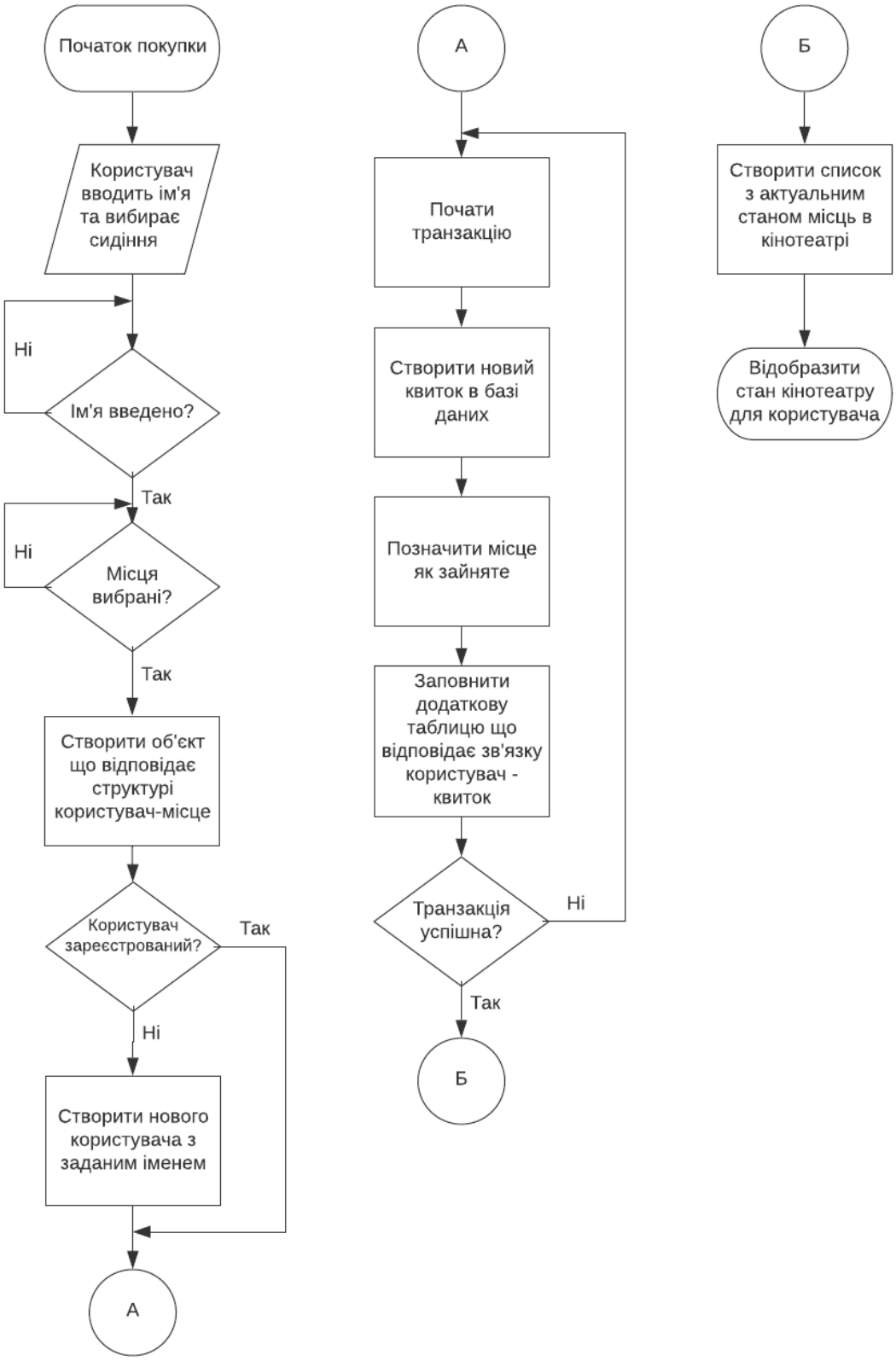
ДОДАТОК В

Клієнт-серверний додаток для керування системою кінотеатру

Принципова схема алгоритму покупки квитків

ІАЛЦ.467400.005 Д3

Аркушів 1



					ІАЛЦ.467400.005 Д3		
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Разробив		*Мілашевський Є. В.*			Клієнт-серверний додаток для керування системою кінотеатру Принципова схема	*Літ.*	*Аркуш* 58 / *Аркушів* 1
Керівник		*Ружевський М.С.*					
Реценз.						*НТУУ "КПІ" ФІОТ* *ІП-62*	
Н. Контр.		*Сімоненко В.П.*					
Затв.							

ДОДАТОК Г
Розподілена система для обробки даних з датчиків

Лістинг частин програми
ІАЛЦ.467400.006 Д4

Аркушів 9

MainController.java

```
package com.studying.cinema.controllers;

@Controller
public class MainController {

    @Value("${movieTheatreName}")
    private String movieTheatreName;

    @Autowired
    private MovieService movieService;
    @Autowired
    private TrailerService trailerService;
    @Autowired
    private NewsService newsService;
    @Autowired
    private PlaceService placeService;
    @Autowired
    private UserService userService;

    @ModelAttribute("movieTheatreName")
    public void addMovieTheatreName(Model model) {
        model.addAttribute("movieTheatreName", movieTheatreName);
    }

    @GetMapping({"/", "index"})
    public String mainPage(Model model) {
        model.addAttribute("mainPoster", movieService.getMainPosterDto());
        model.addAttribute("posters", movieService.getPostersForMainPage());
        model.addAttribute("trailer", trailerService.getNewestTrailer());
        model.addAttribute("news", newsService.getLatestNews(2));
        model.addAttribute("mostPopularMovies", movieService.getMostPopularMovies());
        return "index";
    }

    @GetMapping("/contact")
    public String contact(Model model) {
        model.addAttribute("message", new MessageDto());
        return "contact";
    }

    @GetMapping("/reviews")
    public String reviews(Model model) {

        return "reviews";
    }

    @GetMapping("/booking")
    public String bookSeatsPage(Model model) {
        Map<String, List<PlaceDto>> allPlaceMap = placeService.getAllPlaceMap("Hall-A");
        model.addAttribute("rowNumbers", IntStream.rangeClosed(1,
allPlaceMap.get("A").size()).boxed().collect(Collectors.toList()));
        model.addAttribute("mapPlace", allPlaceMap);
        model.addAttribute("userDto", new UserOrderDto());
        return "seats";
    }

    @PostMapping("/booking")
    public String bookSeatsConfirm(@ModelAttribute UserOrderDto userOrderDto, Model model) {
        userService.updateUserStatus(userOrderDto);
        Map<String, List<PlaceDto>> allPlaceMap = placeService.getAllPlaceMap("Hall-A");
```

```
        model.addAttribute("rowNumbers", IntStream.rangeClosed(1,
allPlaceMap.get("A").size()).boxed().collect(Collectors.toList()));
        model.addAttribute("mapPlace", allPlaceMap);
        model.addAttribute("userDto", new UserOrderDto());
        return "seats";
    }

    @GetMapping("/videos")
    public String videos(Model model,
                         @RequestParam(required = false, defaultValue = "0") int page,
                         @RequestParam(required = false, defaultValue = "12") int count) {
        Page<MoviePosterDto> posterPage = movieService.getPosterPage(page == 0 ? page : page - 1, count);
        List<Integer> pageNumbers = IntStream.rangeClosed(1, (posterPage.getTotalPages() > 0 ?
posterPage.getTotalPages() : 1)).boxed().collect(Collectors.toList());
        model.addAttribute("posterPage", posterPage.getContent());
        model.addAttribute("pageNumberList", pageNumbers);
        return "videos";
    }

    @GetMapping("/single")
    public String single(Model model) {
        return "single";
    }

    @PostMapping("/contact")
    public String postMessage(@ModelAttribute MessageDto message, Model model) {
        System.out.println(message);
        model.addAttribute("message", message);
        model.addAttribute("success", true);
        return "contact";
    }



}
```

MovieService.java

```
package com.studying.cinema.service;

@Service
public class MovieService {
    @Autowired
    private MovieRepository movieRepository;

    @Autowired
    private MovieMapper movieMapper;

    public MovieDto getMainPosterDto() {
        Movie mainPoster = movieRepository.findFirstByMainPosterFlagTrueOrderByReleaseDateDesc();
        return new MovieDto(mainPoster);
    }

    public List<MoviePosterDto> getPostersForMainPage() {
        List<Movie> topMovies = movieRepository.findTop6ByOrderByReleaseDateDesc();
```

```
        return movieMapper.toMoviePosterDtoList(topMovies);
    }

    public List<MoviePosterDto> getMostPopularMovies() {
        List<Movie> topMovies = movieRepository.findTop4ByOrderByRating();
        return movieMapper.toMoviePosterDtoList(topMovies);
    }

    public List<MoviePosterDto> getAllPosters() {
        List<Movie> moviesWithLatestTrailer = movieRepository.findAll();

        return movieMapper.toMoviePosterDtoList(moviesWithLatestTrailer);
    }

    public Page<MoviePosterDto> getPosterPage(int page, int count) {
        PageRequest pageable = PageRequest.of(page, count);
        Page<Movie> moviePage = movieRepository.findAll(pageable);
        return new PageImpl<>(moviePage.stream().map(MoviePosterDto::new).collect(Collectors.toList()),
pageable, moviePage.getTotalElements());
    }
}
```

NewsService.java

```
package com.studying.cinema.service;

@Service
public class NewsService {
    @Autowired
    private NewsRepository newsRepository;

    public List<NewsDto> getLatestNews(int count) {
        Pageable top = PageRequest.of(0, count);
        List<News> news = newsRepository.findAllByOrderByPublishingDateTimeDesc(top);
        List<NewsDto> newsDto = news.stream().map(NewsDto::new).collect(Collectors.toList());
        return newsDto;
    }

}
```

PlaceService.java

```
package com.studying.cinema.service;
```

```
@Service
public class PlaceService {

    @Autowired
    private PlaceRepository placeRepository;

    @Autowired
    private HallRepository hallRepository;


    public Map<String, List<PlaceDto>> getAllPlaceMap(String hallName) {
        Hall hall = hallRepository.findByHallName(hallName);
        List<Place> places = placeRepository.findAllByHallOrderByPlaceLineAscPlaceNumberAsc(hall);
        Map<String, List<PlaceDto>> map = new HashMap<>();
        places.forEach(place -> {
            List<PlaceDto> placeDtos = map.get(place.getPlaceLine());
            if (placeDtos == null) {
                placeDtos = new ArrayList<>();
            }
            placeDtos.add(new PlaceDto(place));
            map.put(place.getPlaceLine(), placeDtos);
        });
        return map;
    }

    public List<PlaceDto> makePlacesUnavailable(List<String> placeKeyList) {
        Map<String, Integer> places = new HashMap<>();
        placeKeyList.forEach(s -> {
            String row = s.substring(0, 1);
            Integer column = Integer.valueOf(s.substring(1));
            places.put(row, column);
        });
        List<Place> placeList = places.entrySet().stream().map(stringIntegerEntry ->
placeRepository.findByPlaceLineAndPlaceNumber(stringIntegerEntry.getKey(),
stringIntegerEntry.getValue())).peek(place -> place.setAvailable(false)).collect(Collectors.toList());
        placeList.forEach(placeRepository::save);
        return placeList.stream().map(PlaceDto::new).collect(Collectors.toList());
    }
}
```

TicketService.java

```
package com.studying.cinema.service;

@Service
public class TicketService {
    @Autowired
    private TicketRepository ticketRepository;
    @Autowired
    private PlaceRepository placeRepository;


    public List<Ticket> createTickets(User user, List<String> seats) {
        List<Pair<String, Integer>> seatList = new ArrayList<>();
        seats.forEach(s -> {
            String row = s.substring(0, 1);
            Integer column = Integer.valueOf(s.substring(1));
            Pair<String, Integer> pair = Pair.of(row, column);
            seatList.add(pair);
        });
        List<Place> placeList = seatList.stream().map(stringIntegerPair ->
placeRepository.findByPlaceLineAndPlaceNumber(stringIntegerPair.getFirst(), stringIntegerPair.getSecond()))
                .peek(place -> place.setAvailable(false)).collect(Collectors.toList());
        placeRepository.saveAll(placeList);
        List<Ticket> tickets = placeList.stream()
                .map(place -> createTicket(user, place))
                .collect(Collectors.toList());
        return ticketRepository.saveAll(tickets);
    }

    public Ticket createTicket(User user, Place place) {
        Ticket ticket = new Ticket();
        ticket.setPlace(place);
        ticket.setUser(user);
        ticket.setPrice(80000);
        return ticket;
    }
}
```

TrailerService.java

```
package com.studying.cinema.service;
```

```
@Service
public class TrailerService {
    @Autowired
    private TrailerRepository trailerRepository;
    @Autowired
    private TrailerMapper trailerMapper;

    public TrailerDto getNewestTrailer() {
        Trailer trailer = trailerRepository.findTopByOrderByUploadDateDesc();
        String movieTitle = trailer.getMovie().getMovieTitle();
        TrailerDto trailerDto = new TrailerDto(movieTitle, trailer.getTrailerLink());
        return trailerDto;
    }

    public List<TrailerDto> getTrailerPageOrderByDate(int page, int count) {
        Pageable trailerPage = PageRequest.of(page, count);
        List<Trailer> trailers = trailerRepository.findLatestTrailerForEveryMovie(trailerPage);
        return trailerMapper.toTrailerDtoList(trailers);
    }
}
```

UserService.java

```
package com.studying.cinema.service;

@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;
    @Autowired
    private TicketService ticketService;

    @Transactional
    public void updateUserStatus(UserOrderDto userOrderDto) {
        User user = userRepository.findByUserName(userOrderDto.getName()).orElseGet(() -> {
            User user1 = new User();
            user1.setUserName(userOrderDto.getName());
            return user1;
        });
        user.setUserName(userOrderDto.getName());
        user.setTickets(ticketService.createTickets(user, userOrderDto.getSeats()));
        userRepository.save(user);
    }
}
```

CinemaApplication.java

```
package com.studying.cinema;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class CinemaApplication {

    public static void main(String[] args) {
        SpringApplication.run(CinemaApplication.class, args);
    }

}
```

MovieRepository.java

```
package com.studying.cinema.model.db.crud.mysql.repositories;

public interface MovieRepository extends JpaRepository<Movie, Long> {
    Movie findFirstByMainPosterFlagTrueOrderByReleaseDateDesc();

    List<Movie> findTop6ByOrderByReleaseDateDesc();

    List<Movie> findTop4ByOrderByRating();

    Page<Movie> findAll(Pageable pageable);
}
```

HallRepository.java

```
package com.studying.cinema.model.db.crud.mysql.repositories;

import com.studying.cinema.model.db.crud.mysql.entities.Hall;
import org.springframework.data.jpa.repository.JpaRepository;

public interface HallRepository extends JpaRepository<Hall, Long> {
    Hall findByHallName(String hallName);
}
```

NewsRepository.java

```
package com.studying.cinema.model.db.crud.mysql.repositories;

import com.studying.cinema.model.db.crud.mysql.entities.News;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface NewsRepository extends JpaRepository<News, Long> {
    List<News> findAllByOrderByPublishingDateTimeDesc(Pageable pageable);
}
```

PlaceRepository.java

```
package com.studying.cinema.model.db.crud.mysql.repositories;


public interface PlaceRepository extends JpaRepository<Place, Long> {
    List<Place> findAllByHallOrderByPlaceLineAscPlaceNumberAsc(Hall hall);

    Place findByPlaceLineAndPlaceNumber(String placeLine, Integer placeNumber);


}
```

TrailerRepository.java

```
package com.studying.cinema.model.db.crud.mysql.repositories;

import com.studying.cinema.model.db.crud.mysql.entities.Trailer;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

import java.util.List;

public interface TrailerRepository extends JpaRepository<Trailer, Long> {

   Trailer findTopByOrderByUploadDateDesc();

   List<Trailer> findAllByOrderByMovieAscUploadDateDesc(Pageable pageable);

   @Query("select t from Trailer t " +
         "where t.uploadDate in (select max(tt.uploadDate) from Trailer tt group by tt.movie ) " +
         "order by t.uploadDate desc ")
   List<Trailer> findLatestTrailerForEveryMovie(Pageable pageable);

}
```

Hall.java

```
package com.studying.cinema.model.db.crud.mysql.entities;

import lombok.Getter;
import lombok.Setter;

import javax.persistence.*;
import java.util.List;

@Entity
@Getter
@Setter
public class Hall {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private long idHall;
   private String hallName;
   private boolean state;
   @OneToMany(cascade = CascadeType.REMOVE, mappedBy = "hall")
   private List<Session> sessions;
   @ManyToOne(fetch = FetchType.LAZY)
   @JoinColumn(name = "idMovieTheatre")
   private MovieTheatre movieTheatre;
   @OneToMany(cascade = CascadeType.REMOVE, mappedBy = "hall")
   private List<Place> places;



}
```

Movie.java

```
package com.studying.cinema.model.db.crud.mysql.entities;

import lombok.Data;

import javax.persistence.*;
import java.time.LocalDate;
import java.util.List;
import java.util.Set;
```

```
@Entity
@Data
public class Movie {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long idMovie;
    private String movieTitle;
    private LocalDate releaseDate;
    private String description;
    private String ageLimit;
    private Long duration;
    private int rating;
    private String pictureLink;
    private boolean mainPosterFlag;
    @ManyToMany(mappedBy = "movies")
    private List<MovieGenre> genres;
    @OneToMany(cascade = CascadeType.REMOVE, mappedBy = "movie")
    private List<Trailer> trailers;
    @ManyToMany(mappedBy = "movies")
    private List<MovieDirector> creators;
    @OneToMany(mappedBy = "movie")
    private List<Session> session;


}
```